WL-TR-93-3096

SOFTWARE USER'S MANUAL FOR STABILITY AND TRANSITION ANALYSIS WITH THE CODES LSH AND PSH

TH. HERBERT
G. K. STUCKERT
N. LIN

DYNAFLOW, INC.
3040 RIVERSIDE DRIVE
COLUMBUS OH 43221

September 1993

Final Report for 09/27/90 - 09/27/93

Approved for public release; distribution is unlimited

FLIGHT DYNAMICS DIRECTORATE
WRIGHT LABORATORY
AIR FORCE MATERIEL COMMAND
WRIGHT-PATTERSON AFB OH 45433-7562

NOTICE

When Government drawings, specifications, or other data are used for any purpose other than in connection with a definitely Government-related procurement, the United States Government incurs no responsibility or any obligation whatsoever. The fact that the government may have formulated or in any way supplied the said drawings, specifications, or other data, is not to be regarded by implication, or otherwise in any manner construed, as licensing the holder, or any other person or corporation; or as conveying any rights or permission to manufacture, use, or sell any patented invention that may in any way be related thereto.

This report is releasable to the National Technical Information Service (NTIS). At NTISS, it will be available to the general public, including foreign nations.

This technical report has been reviewed and is approved for publication.

Roger Kimmel

Roger Kimmel, Program Monitor
Aerothermodynamic Res Section
Aerothermodynamics & Flight Mech
Research Branch

Valentine Dahlem

Valentine Dahlem, Chief
Aerothermodynamics & Flight Mech
Research Branch

Dennis Sedlock

Dennis Sedlock, Actg Chief
Aeromechanics Division

If your address has changed, if you wish to be removed from our mailing list, or if the addressee is no longer employed by your organization, please notify WL/FIMH, WPAFB, OH 45433-7936 to help us maintain a current mailing list.

Copies of this report should not be returned unless return is required by security considerations, contractual obligations, or notice on a specific document.

REPORT DOCUMENTATION PAGE

Form Approved
OMB No. 0704-0188

1. AGENCY USE ONLY *(Leave blank)*	2. REPORT DATE	3. REPORT TYPE AND DATES COVERED
	SEP 1993	FINAL 09/27/90--09/27/93

4. TITLE AND SUBTITLE SOFTWARE USER'S MANUAL FOR STABILITY AND TRANSITION ANALYSIS WITH THE CODES LSH AND PSH

5. FUNDING NUMBERS
C F33615-90-C-3009
PE 62201
PR 2404
TA 07
WU B3

6. AUTHOR(S)
TH. HERBERT
G. K. Stuckert
N. Lin

7. PERFORMING ORGANIZATION NAME(S) AND ADDRESS(ES)
DYNAFLOW, INC.
3040 RIVERSIDE DR.
COLUMBUS OH 43221

8. PERFORMING ORGANIZATION REPORT NUMBER

9. SPONSORING / MONITORING AGENCY NAME(S) AND ADDRESS(ES)
FLIGHT DYNAMICS DIRECTORATE
WRIGHT LABORATORY
AIR FORCE MATERIEL COMMAND
WRIGHT PATTERSON AFB OH 45433-7562

10. SPONSORING / MONITORING AGENCY REPORT NUMBER
WL-TR-93-3096

11. SUPPLEMENTARY NOTES
THIS IS A SMALL BUSINESS INNOVATION RESEARCH REPORT, PHASE 2

12a. DISTRIBUTION / AVAILABILITY STATEMENT
APPROVED FOR PUBLIC RELEASE; DISTRIBUTION UNLIMITED

12b. DISTRIBUTION CODE

13. ABSTRACT *(Maximum 200 words)*

This manual describes how to use the local linear stability code LSH and the nonlinear Parabolized Stability Equations (PSE) solver PSH for compressible flows. Both codes are adaptable to analysis of different flows over fairly general shapes of bodies. For analysis of a new problem, the user may specify the basic state, the coordinate system, dependent distrubance variables and their boundary conditions to be used for the stability analysis through physics insert files. Once these preliminary steps are completed, different tasks of stability analysis and PSE calaculations can be carried out by simple modifications of a few input files. Both codes are highly modular and closely integrated. Output data from LSH can be used directly as initial data for PSH runs. Examples of several insert files and input files of the built-in problems are given to assist the user in setting up a new problem.

14. SUBJECT TERMS
LINEAR STABILITY EQUATIONS
PARABOLIZED STABILITY EQUATIONS
COMPRESSIBLE FLOWS
GENERAL BODIES

15. NUMBER OF PAGES
84

16. PRICE CODE

17. SECURITY CLASSIFICATION OF REPORT	18. SECURITY CLASSIFICATION OF THIS PAGE	19. SECURITY CLASSIFICATION OF ABSTRACT	20. LIMITATION OF ABSTRACT
UNCLASSIFIED	UNCLASSIFIED	UNCLASSIFIED	UL

Contents

List of Figures

Chapter 1

Overview

1.1 What is LSH

LSH is a code for analyzing the linear stability of basic states. The main code is generic and void of any particular physical problem yet provides for various tasks that frequently occur in the stability analysis of fluid motions:

global	— eigenvalue spectra,
local	— single eigenvalues and eigenfunctions,
table	— one- or more-dimensional tables of eigenvalues,
curve	— (neutral) curves in parameter space,

and others. A specific physical problem can be defined in a set of files that describe the basic state and the stability equations. These files are included at compile time, while definitions, tasks, and parameters are read from two additional files during run time.

The original version C-S of LISA[1] was developed to be able to obtain some additional results on some stability problem in a few minutes if the insert files specific to this problem are available, and to obtain the first results for a new problem within a few hours if the equations are available (see LISA Version C-S 2.0 User's Manual). Since the specifications for a new problem are usually short, the debugging effort is greatly reduced. Often, the specification of a new case can be eased by referring to the basic state or the stability equations of an existing case through the *Makefile*. This Version C-S is based on a spectral collocation method that has proven reliable and able to provide highly accurate results for a wide spectrum of applications.

The present Version WL of LSH is a derivative of LISA and has a similar modular structure, but is tailored for *compressible* flows over aerodynamic bodies with curvature. Unlike the version C-S, the stability equations here are not specified by the user. They are already coded for three-dimensional disturbances, derived from the

[1] LISA is a proprietary software product of DynaFlow, Inc. for the commercial market.

Navier-Stokes equations for compressible flows. The analysis is done in general curvilinear coordinates defined by the user. The basic flow to be studied is supplied by the user either by analytical calculation or reading in from a data set. The user may also specify boundary conditions for disturbances and choose thermodynamic properties and dependent variables for disturbances. Hence, the code is highly modular and adaptable to different problems. The stability equations in this version are discretized by a high-order hermitian finite-difference method. This method is accurate yet more efficient than the spectral method.

1.2 What is PSH

PSH is the code for solving the compressible PSE for three-dimensional linear or nonlinear disturbances. The compressible PSE including nonlinear terms are formulated in general curvilinear coordinates and are already coded (Stuckert & Herbert, 1993). The user should refer to the final report "Methods for Transition Prediction in High-speed Fows" for details of the equations. The rest of the physics of the problem is specified by the user by *insert* files in the same manner as for LSH. In fact, most of the generic program modules, insert files and an input file `Definitions` are common to PSH and LSH. These physics files are described in detail in Section 3. LSH solves an eigenvalue problem or a series of eigenvalue problems. Initial conditions for PSH are obtained from the local solution(s) obtained with LSH. The PSH then integrates the PSE in the parabolic marching direction, solving the system (in general, nonlinear) of equations at each marching step. With PSH, linear and nonlinear developments of disturbances, interaction between different modes of disturbance can be followed close to the beginning point of transition. This location is usually manifested by the minimum in mean skin friction and its rapid rise thereafter, etc. Hence, given an accurate model of initial data, transition can be predicted without any empirical data. The computing effort required for PSH is much less than that for Direct Navier-Stokes simulations (DNS).

1.3 The Program

The codes LSH and PSH are highly modular. They are structured as shown in Figure 1. For this Version WL 1.0, the graphical user interface (GUI) is under development for Silicon Graphics workstations under IRIX 4.0.5 or higher. Other platforms will follow. The batch versions have been tested in various Unix environments and can be adapted to other systems by minor modifications, especially of the *Makefiles*. The codes are partly written in C to provide the connection to the graphical user interface and for more efficient memory management, input, and output. Other parts, including the interface to the user, are written in Fortran 77 and use the C preprocessor during object-code generation. The following describes only the batch versions of LSH and

PSH.

The main codes are provided as two object files, `lsh.o` and `psh.o`, and a library archive `libut.a`. The library contains the eigenvalue solvers *RG* and *CG* of EISPACK, some routines of LINPACK, some other linear algebra solvers, and the routines associated with the spectral collocation method. The linear algebra routines consume most of the CPU time. All floating-point operations are performed in double precision, real*8 or complex*16.

In spite of some lacking features, Fortran 77 is widely available and used in engineering applications. The goal to keep the code generic and to write the problem-oriented "physics insert files" in Fortran 77 required compromises. Especially the Fortran hierarchy of statements (which prohibits, e.g., dimension after executable statements) made the placement and structure of the "physics insert files" mandatory.

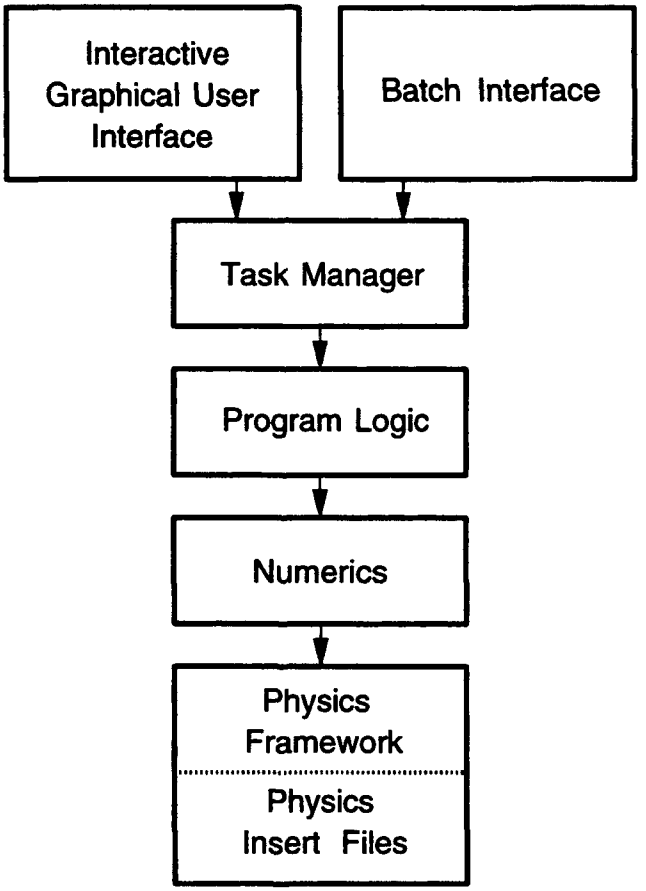

Figure 1.1: Program Structure for LSH and PSH

Chapter 2

The Physics Files

Various data must be specified to adapt the generic code to a specific problem. This specification is made through two groups of files: **insert files** and **input files**. There are eight **insert files** with suffix `.ins` used by both LSH and PSH, i.e.,

```
common.ins
gridbs.ins
metric.ins
bstate.ins
btrans.ins
boundary.ins
pointd.ins
```

and two **insert files** with suffix `.ins` which are used by PSH for postprocessing:

```
postpr.ins
plotps.ins
```

As for **input files**, there is one common to both LSH and PSH:

```
Definitions
```

one **input file** specific for LSH:

```
Parameters
```

and two **input files** specific for PSH:

```
Control
Modes
```

The insert files are included into the executable code. The input files are read during run time. The input file **Modes** which provide initial station data for PSH is created by running LSH at the desired station with proper parameters. For the applications that come with the code, the user has to deal only with the problem-specific input files. The insert files are already prepared. For new applications, the developer has to prepare the insert files and the input files. This preparation usually consists of coding the calculation of the basic state in `bstate.ins` and filling the proper array elements in `metric.ins`, `btrans.ins`, etc.

2.1 Directory Structure and Makefiles

Before we discuss preparation and contents of the physics files, it is useful to consider the arrangement of files in directories and how the executable codes for a specific stability are built. Under the main directory *w.psh* there are numerous directories. The first character of the directory name characterizes the contents:

*w.**: general objects for LSH and PSH
*c.**: complex problems
*b.**: basic states; `gridbs.ins`, `bstate.ins`, `btrans.ins`
*e.**: stability equations; `common.ins`, `metric.ins`, etc.

All specific problems are in separate directories *c.**. Each of these directories contains at least four files:

```
Makefile
Definitions
Parameters
Control
```

Initially, there is no executable code `lsh` or `psh` in this directory. The code has to be made using the Unix command

```
$ make
```

By default, this command executes the instructions in the `Makefile`. The first line of the `Makefile` defines two "macros":

BSTATE	= directory which contains `gridbs.ins`, `bstate.ins`, and `btrans.ins`;
VECTOR	= directory which contains `common.ins` , `metric.ins`, `boundary.ins`, and `pointd.ins`.

If either set of files, say `bstate.ins`, is located in the present working directory *c.case*, then either `BSTATE=.../c.case` or `BSTATE=`*'pwd'* would be appropriate directives. In the latter form, the left quotes request that the system set `BSTATE` to the output of the *pwd* command. This latter form is more generic, because it is independent of the current directory.

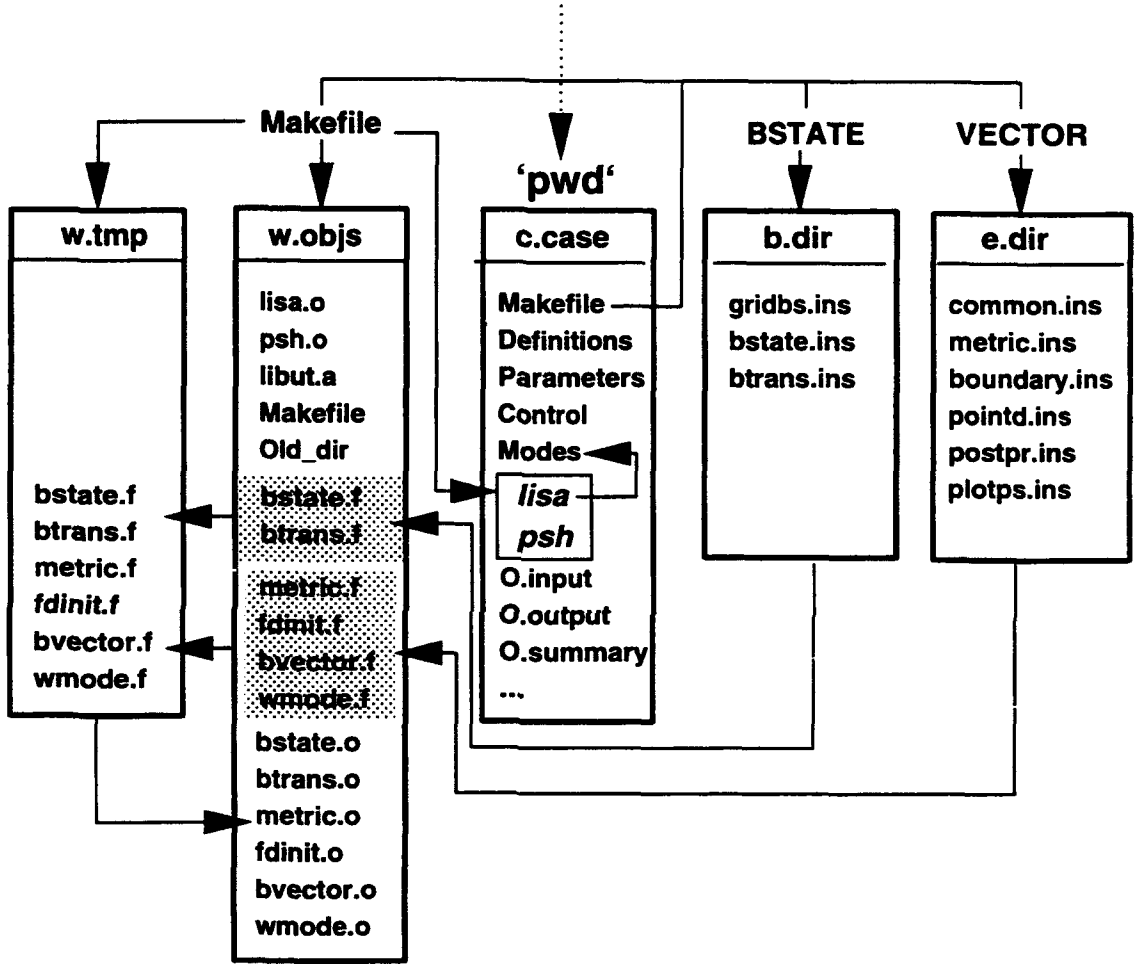

Figure 2.1: Directory Structure and Makefiles for LSH and PSH

Otherwise, the `Makefile` leaves the task of making the executable code to the `../Makefile` in the parent directory *w.psh*. Figure 2.1 illustrates how the Makefiles interact with the directories associated with the problem in *c.case*. In essence, the physics insert files are read from their directories, included into the fortran `*.f` files in *w.objs*, and stored in *w.tmp*. The completed files in *w.tmp* are compiled into object files `*.o` in *w.objs* and combined with `lsh.o`, `psh.o` and the library `libut.a` to produce `lsh` and `psh` in the current working directory. The executable codes `lsh` and `psh` will remain there even if the user works on other problems, until the file is removed with the Unix command `rm` or

```
$ make clean
```

which also cleans the problem-specific object files in *w.objs* and prompts for the removal of the output files.

Since **make** remakes objects only if they are out of date, the file `Old_dir` in *w.objs* serves to register if the user changes to another problem with different insert files. The executable code **lsh** or **psh** is remade if the insert files change.

The flexibility in accommodating the insert files in the same or different directories avoids unnecessary duplications. Often it is only convenient to protect the insert files from unintentional damage in a heavily used directory. New applications must be prepared in directories on the same level, i.e., in directories *w.psh/c.new-appl* to be compatible with the various Makefiles.

2.2 Preparing the Insert Files

In the following, we discuss the details of the insert files. This section is primarily provided for developers of new applications. All insert files have access to the following shared information:

`zro`	$= 0$, the real double precision number zero
`one`	$= 1$, the real double precision number one
`zero`	$= (0,0)$, the complex double precision number zero
`uimag`	$= (0,1)$, the complex double precision imaginary unit
`pi`	the real double precision number $\pi = 3.1415\cdots$.
`jx`	the highest index of the collocation points, $j = 0,\cdots,jx$.
`nx`	the highest order of differentiation.
`ncon`	the number of constants or zero (if there are no constants).
`npar`	the number of parameters.
`nequ`	the number of equations in the stability equations.
`nvar`	the number of variables $nvar = nequ$ in the problem.
`isize`	the dimension $(jx+1)\cdot nequ$ of the algebraic system.
`con(i)`	the constants C_i for $i = 1,\cdots,ncon$.
`par(i)`	the parameters P_k for $k = 1,\cdots,npar$.
`chy(j)`	the collocation points ξ_j^2, $j = 0,\cdots,jx$.

The following variables related to **PSH** are also available.

`modt`	the highest index of the temporal frequency, the lowest index is 0.
`modzm`	the lowest index of wavenumber in ξ^3 direction
`modzp`	the highest index of wavenumber in ξ^3 direction
`izsym`	integer indicating symmetry ξ^3 direction for modes

`parm(k,it,iz,l)`	the parameters P_k for $k = 1,...,npar$; and modes $it = 0,...,modt$; $iz = (1 - izsym) * modzm,...,modzp$ at $l = lprev$ or $l = lcur$ step.
`yofx(j,l)`	the collocation points ξ_j^2, $j = 0,\cdots,jx$ at $l = lcur$ or $l = lprev$ step.
`dydx(j)`	$\partial\xi^2/\partial\xi_j^1$, at $j = 0,\cdots,jx$ at the current step.

The variable names follow the Fortran convention that all names beginning with `i-n` are implicitly integer. All variables with names beginning with `a-h` or `o-z` are implicitly declared double precision, `real*8`. Most of these variables are taken from the input files `Definitions`, `Parameters`, or `Control`.

2.2.1 common.ins

This file is for communication between insert files `bstate.ins`, `btrans.ins`, `metric.ins`, etc., and the stability equations of the main code. The file allocates *local* arrays for all basic state quantities, metric terms and declares all complex variables, used in the stability equations. The maximum number of basic-state collocation points per station to be read in, `Fortran` I/O unit numbers for input files associated with basic state and user-written output files are also specified with `parameter` statements. The specification of `jmax` is independent of `jx` used for stability analysis.

No executable statements are allowed in `common.ins`.

2.2.2 gridbs.ins

Subroutine `gridbs` generates the grid for the basic state calculation. It usually does this simply by reading in the grid of a previously computed basic state.

For instance, if the basic state is computed with a separate boundary layer code, PNS code, or Navier-Stokes code, then `gridbs` must read in the corresponding grid. This grid should then be made available through a common block to subroutine `bstate` which will have to interpolate the solution onto the *disturbance* grid.

If, however, the basic state solution is analytical or can be otherwise easily computed, the `chy(j)` should be used as the basic state grid. In this case, `gridbs` is redundant, and the file `gridbs.ins` may be empty since `chy(j)` is directly available to `bstate`.

Subroutine `gridbs` may also rescale the basic state grid and solution with the disturbance scales, or vice versa, and perform other necessary basic state initializations.

Like `bstate.ins` in `bstate.f`, `gridbs.ins` is included in `gridbs.f` before any excutable statement. Hence, the user may declare new arrays as needed.

2.2.3 metric.ins

The coordinate system chosen for the stability analysis must be specified by the user by assigning values to the covariant basis vectors $\partial x^i/\partial \xi^j$ and their first and second partial derivatives with respect to $\partial \xi^l$ in the file `metric.ins`. Here x^i is the i^{th} Cartesian coordinate and ξ^j is the j^{th} curvilinear coordinate used in the stability analysis. Hence, the following corresponding arrays are defined in this insert file:

`dxdc(j,i)` $= \partial x^i/\partial \xi^j$, covariant basis vectors $\xi^j, j = 1, nd$ where nd is the number of dimensions . nd is 4 usually, with 3 spatial coordinates and a fourth temporal coordinate.

`d2xdc2(k,j,i)` $= \partial^2 x^i/\partial \xi^j \partial \xi^k$, first partial derivatives of covariant basis vectors with respect to ξ^k; $k = 1, nd$.

`d3xdc3(l,k,j,i)` $= \partial^3 x^i/\partial \xi^j \partial \xi^k \partial \xi^l$, second partial derivatives of covariant basis vectors with respect to ξ^k and ξ^l, $k = 1, nd$;, $l = 1, nd$.

Symmetry of metric terms are used in the subroutine `metric` so that the user needs to specify only unique entries of arrays `d2xdc2` and `d3xdc3`. The array `d3xdc3` needs not be specified if Cartesian velocity components are used as dependent variables for stability analysis (`mvar=0` is specified). Subroutine `metric` uses these in `metric.ins` to compute the terms $\partial \xi^i/\partial x^j$, $\partial^2 \xi^i/\partial x^j \partial x^k$, and $\partial^3 \xi^i/\partial x^j \partial x^k \partial x^l$ which appear in the stability equations after transforming the x^i derivatives of the disturbances to ξ^j derivatives with the chain rule. It is important to note that the approximations to the stability equations (either local or PSE) are made after the coordinate transformation. Hence, the coordinate system chosen will have some effect on the accuracy of the results obtained.

For each streamwise station ξ^1, these covariant and contravariant metric terms for all (jx) normal grid points are saved in global arrays. These arrays are used in other modules of the code when necessary. For every $n-th$ collocation point at a station, the following metric terms are available for the user, e.g., to postprocess the station solution in the insert files `postpr.ins`.

$$\texttt{dxdca(n,j,i)} = \frac{\partial x^i}{\partial \xi^j}$$

$$\texttt{d2xdca(n,ks,i)} = \frac{\partial^2 x^i}{\partial \xi^j \partial \xi^k}$$

$$\texttt{d3xdca(n,ls,i)} = \frac{\partial^2 x^i}{\partial \xi^j \partial \xi^k \partial \xi^l}$$

$$\texttt{gda(n)} = \sqrt{\frac{\partial x^k}{\partial \xi^i}\frac{\partial x^k}{\partial \xi^i}}$$

$$
\begin{aligned}
\texttt{sqrtga(n)} &= \sqrt{\frac{1}{\frac{\partial x^k}{\partial \xi^i}\frac{\partial x^j}{\partial \xi^i}}} \\
&= Jacobian \\
\texttt{dcdxa(n,j,i)} &= \frac{\partial \xi^i}{\partial x^j} \\
\texttt{d2cdxa(n,ks,i)} &= \frac{\partial^2 \xi^i}{\partial x^j \partial x^k} \\
\texttt{d3cdxa(n,ls,i)} &= \frac{\partial^2 \xi^i}{\partial x^j \partial x^k \partial x^l}
\end{aligned}
$$

$$i = 1, ..., nd$$

$$((((ls = ls + l),\ l = 1, ..., k,\ \ ks = ks + k),\ k = 1, ..., j),\ j = 1, ..., 4)$$

2.2.4 btrans.ins

In stability analysis, the equations are derived for disturbances, $\tilde{Q}^i$, $i = 1, ..., nvar$ about a laminar basic flow, Q^i. The stability code requires that basic state, in variables Q^i be specified for each grid point in the stability analysis coordinate system ξ^j specified in `metric.ins`. The basic state is usually obtained in the insert file `gridbs.ins` `bstate.ins` in variables and a coordinate system suitable for its solution. Instead of requiring the user to transform these into Q^i in stability coordinates directly, more flexibility is provided by using the subroutine `btrans.f` which allows specification of three sets of variables through insert file `btrans.ins`. Note that subroutine `btrans` has executable statements before including `btrans.ins`. Therefore, *no* array declarations should be made in this insert file. The insert file `btrans.ins` must be coded so that it performs the following tasks.

1. The variables defined in `bstate.ins` are transformed into density and temperature as the first $nts = 2$ dependent variables and one of the following sets of velocity components as $nts + 1$ to $nvar$ variables:

   (a) Cartesian velocity components.

   (b) Physical contravariant velocity components in the stability coordinate.

   (c) Contravariant velocity components in the stability coordinate.

   The user must specify one of the above three sets by setting `ibvel` to either 0, 1, or 2, respectively.

2. The derivatives of the variables obtained in `bstate.ins` are transformed into derivatives with respect to curvilinear stability coordinates ξ^j.

3. The derivatives of the equation of state and specific enthalpy with respect to thermodynamic dependent variables are assigned.

Hence, the following arrays of basic state variables are to be defined by the user in this insert file.

`bsoln(i)` $= Q^i$, i-th dependent variable for basic state, with $i = 1$ to nts representing thermodynamic dependent variables ρ and T, and $nts + 1$ to $nvar$ Cartesian, physical contravariant or contravariant velocity components.

`dbdc(j,i)` $= \partial Q^i / \partial \xi^j$, first partial derivatives of Q^i with respect to curvilinear coordinates, $\xi^j, j = 1, nd$ where nd is the number of spatial dimensions, usually 3.

`d2bdc2(k,j,i)` $= \partial^2 Q^i / \partial \xi^j \partial \xi^k$, second partial derivatives of basic state with respect to ξ^j and ξ^k, $j = 1, nd$; $k = 1, nd$.

`ddq1p(i)`, `ddq2p(i,j)`, ddq3p(i,j,k), first, second, and third partial derivatives of the equation of state $p(\rho, T)$ with respect to thermodynamic variables, Q^i, $i = 1, nts$, i.e.,

$$\frac{\partial p}{\partial Q^i}, \quad \frac{\partial^2 p}{\partial Q^j \partial Q^i}, \quad \frac{\partial^3 p}{\partial Q^k \partial Q^j \partial Q^i}; \; i, j, k = 1, nts.$$

For example, for a perfect gas, $p = \rho T / \gamma M_\infty^2$. Hence

$$\begin{aligned}
\texttt{ddq1p(1)} &= \frac{\partial p}{\partial \rho} = \frac{T}{\gamma M_\infty^2} \\
\texttt{ddq1p(2)} &= \frac{\partial p}{\partial T} = \frac{\rho}{\gamma M_\infty^2} \\
\texttt{ddq2p(1,1)} &= 0 \\
\texttt{ddq2p(1,2)} &= \texttt{ddq2p(2,1)} = \frac{1}{\gamma M_\infty^2} \\
\texttt{ddq2p(2,2)} &= 0 \; \textit{etc.}
\end{aligned}$$

`ddq1h(i)`, `ddq2h(i,j)`, `ddq3h(i,j,k)`, first, second, and third partial derivatives of specific enthalpy $h(\rho, T)$ with respect to thermodynamic variables , i.e.,

$$\frac{\partial h}{\partial Q^i}, \quad \frac{\partial^2 h}{\partial Q^j \partial Q^i}, \quad \frac{\partial^3 h}{\partial Q^k \partial Q^j \partial Q^i}; \; i, j, k = 1, nts.$$

`ddq1m1(i)`, `ddq2m1(i,j)`, first and second partial derivatives of first coefficient of viscosity μ_1 with respect to thermodynamic variables , i.e.,

$$\frac{\partial \mu_1}{\partial Q^i}, \ \frac{\partial^2 \mu_1}{\partial Q^j \partial Q^i}, \ i,j = 1, nts.$$

`ddq1m2(i)`, `ddq2m2(i,j)`, first and second partial derivatives of second coefficient of viscosity μ_2 with respect to thermodynamic variables, i.e. ,

$$\frac{\partial \mu_2}{\partial Q^i}, \ \frac{\partial^2 \mu_2}{\partial Q^j \partial Q^i}, \ i,j = 1, nts.$$

`ddq1k(i)`, `ddq2k(i,j)`, first and second partial derivatives of thermal conductivity κ with respect to thermodynamic variables, i.e.,

$$\frac{\partial \kappa}{\partial Q^i}, \ \frac{\partial^2 \kappa}{\partial Q^j \partial Q^i}, \ i,j = 1, nts.$$

Higher order partial derivatives of thermodynamic coefficients with respect to temperature. These are needed for evaluating nonlinear terms.

`d3mu1dt`	=	$\frac{\partial^3 \mu_1}{\partial T^3}$
`d4mu1dt`	=	$\frac{\partial^4 \mu_1}{\partial T^4}$
`d3kdt`	=	$\frac{\partial^3 \kappa}{\partial T^3}$
`d4kdt`	=	$\frac{\partial^4 \kappa}{\partial T^4}$
`d4hdt`	=	$\frac{\partial^4 h}{\partial T^4}$
`d5hdt`	=	$\frac{\partial^5 h}{\partial T^5}$

Note that the names of variables here do not necessarily coincide with those used in `bstate.ins` for the basic flow solution. Assignment statements in `btrans.ins` should translate the names of the latter into those of the former. For example, `bstate.ins` might compute the similarity solution for the compressible boundary layer and have as output the streamfunction and its derivatives with respect to similarity coordinates. The user would, thus, transform the similarity variables into one

set of velocity components (described above) in the file `btrans.ins` and transform derivatives with respect to similarity coordinates into those with respect to curvilinear stability coordinates. [1]

Using these user specified basic state values and metric terms, the subroutine `btrans` calculates the following quantities in *Cartesian* coordinates and stores them in global arrays. They are available at all n collocation points for use in the insert files.

$$\begin{aligned}
\texttt{bsolna(n,i)} &= Q^i = \{\rho, T, u, v, w\}^i \\
\texttt{dbdxa(n,j,i)} &= \frac{\partial Q^i}{\partial x^j} \\
\texttt{d2bdxa(n,ls,i)} &= \frac{\partial^2 Q^i}{\partial x^j \partial x^k} \\
\texttt{blapa(n,i)} &= \Delta Q \\
\texttt{bgrddv(n,j)} &= \frac{\partial\, \partial u^i \partial x^i}{\partial x^j}
\end{aligned}$$

The following thermodynamic quantities are also available at all n collocation points.

$$\begin{aligned}
\texttt{thermka(n)} &= \kappa \\
\texttt{amu1(n)} &= \mu_1 \\
\texttt{amu2(n)} &= \mu_2 \\
\texttt{d1qk(n,i)} &= \frac{\partial \kappa}{\partial Q^i}, \\
\texttt{d2qk(n,ls)} &= \frac{\partial^2 \kappa}{\partial Q^i \partial Q^j}, \\
\texttt{d1qmu1(n,i)} &= \frac{\partial \mu_1}{\partial Q^i}, \\
\texttt{d2qmu1(n,ls)} &= \frac{\partial^2 \mu_1}{\partial Q^i \partial Q^j}, \\
\texttt{d1qmu2(n,i)} &= \frac{\partial \mu_2}{\partial Q^i}, \\
\texttt{d2qmu2(n,ls)} &= \frac{\partial^2 \mu_2}{\partial Q^i \partial Q^j}, \\
\texttt{d1qh(n,i)} &= \frac{\partial h}{\partial Q^i}, \\
\texttt{d2qh(n,ls)} &= \frac{\partial^2 h}{\partial Q^i \partial Q^j},
\end{aligned}$$

[1] One might as well do this directly in `bstate.ins`. However, this preprocessing (translation) of the basic state is often independent of the basic state solution process. With this separate translation, it is easier to modify the basic state numerics when necessary.

$$
\begin{aligned}
\texttt{d3qh(n,ks)} &= \frac{\partial^3 h}{\partial Q^i \partial Q^j \partial Q^k}, \\
\texttt{d1q\_thermo(n,i)} &= \frac{\partial \{state\ equation\}}{\partial Q^i}, \\
\texttt{d2q\_thermo(n,ls)} &= \frac{\partial^2 \{state\ equation\}}{\partial Q^i \partial Q^j}, \\
\texttt{d3q\_thermo(n,ks)} &= \frac{\partial^3 \{state\ equation\}}{\partial Q^i \partial Q^j \partial Q^k},
\end{aligned}
$$

$$((((ks = 0, ..., ks + k),\ k = 1, ..., j;\ ls = 0, ..., ls + j),\ j = 1, ..., i),\ i = 1, ..., nts)$$

Note that second and higher derivatives are symmetric, i.e.,

$$\frac{\partial^2 \{\}}{\partial Q^1 \partial Q^2} = \frac{\partial^2 \{\}}{\partial Q^2 \partial Q^1},\ etc.$$

Hence, only unique entries of the derivatives are stored in these arrays.

2.2.5 pointd.ins

After the coordinate system has been specified, the grid point distribution for the stability analysis must be chosen. This is done partly by specifying the choice of stretching function `idtrf`, and parameters (`strfp(i), i=1,...,4`) in the input file `Definitions` and partly through the insert file, `pointd.ins`. The stability grid is given as $\xi^2(j, \xi^1) = \xi^2_{ref}(j) * g(\xi^1)$. Different choices of `idtrf` described in Section 2.3 specify $\xi^2_{ref}(j)$. The insert file `pointd.ins` is included in the subroutine `fdinit` where the stretching function is evaluated. In `pointd.ins`, the user may specify how the grid point distribution in the ξ^2 direction changes with distance downstream, i.e. $g(\xi^1)$.[1] For example, the location of the outer boundary, ξ^2_{max} (stored in the local variable `btrf`) may be varied as a function of the streamwise distance ξ^1 to prescribe:

$$\xi^2(j, \xi^1) = \xi^2_{ref} \sqrt{\frac{\xi^1}{\xi^1_{ref}}}$$

The value of ξ^2_{max} at the reference station is input as `strfp(2)`. If `pointd.ins` is empty, then the same grid point distribution in ξ^2 at reference location will be used for the analysis.

[1] Here, different choices do *not* affect the accuracy of the approximations made to the stability equations, but *do* affect the numerical accuracy with which the discretized equations are solved.

2.2.6 boundary.ins

In this insert file, boundary conditions for disturbances are specified. This insert file is included in the subroutine `bvector`. The user can choose the dependent variables $\boldsymbol{f}$ for the stability analysis by using the flag `mvar` in the input file `Definitions`. The default variables ($\tilde{\boldsymbol{Q}}$, `mvar=0`) are $\tilde{\rho}$, $\tilde{T}$, $\tilde{u}^i$, where $\tilde{\rho}$ is the density disturbance, $\tilde{T}$ is the temperature disturbance, and $\tilde{u}^i$ is the disturbance in the i-th *Cartesian* component of the disturbance velocity. Transformation from $\boldsymbol{f}$ to $\tilde{\boldsymbol{Q}}$ is described by the following equation.

$$\tilde{\boldsymbol{Q}} = \boldsymbol{M}\,\boldsymbol{f} \text{ or } \tilde{Q}^i = M^i_j f^j$$

This matrix $\boldsymbol{M}$ and its derivatives with respect to Cartesian coordinates, for a specified value of `mvar`, are evaluated in the subroutine `setvrm`. They are available in the following global *real* arrays at $n = 0, ..., jx$ collocation points for the user.

$$\begin{aligned}
\texttt{vrma(n,j,i)} &= M^i_j \; ; i = 1, \ldots, nvar, j = 1, \ldots, nvar \\
\texttt{d1xvra(n,k,j,i)} &= \frac{\partial M^i_j}{\partial \xi^k} \; ; k = 1, ..., 4 \\
\texttt{d2xvra(n,ks,j,i)} &= \frac{\partial^2 M^i_j}{\partial \xi^k \partial \xi^l} \; ; (((ks = 0, ks + l), l = 1, \ldots, k), k = 1, \ldots, 3)
\end{aligned}$$

Global arrays for metric terms and basic state (as described in `metric.ins` and `btrans.ins`) are also available here.

The boundary conditions are applied on the variables $\boldsymbol{f}$ chosen for stability analysis. In general, the boundary condition for the i-th equation is imposed by specifying the `real` array `bvec(i,m,l,k)` in the boundary-condition equation

$$\texttt{bvec(i,m,l,k)} \frac{\partial^m f^l}{\partial \xi^{2,m}} = \texttt{bvrhs(i,k)} \; ; l = 1, \ldots, nvar, i = 1, \ldots, nequ$$

Here, $m = 0, ..., nx$, $k = 0$ or 1 for the boundary condition at $\xi^2 = \xi^2_{min}$ or $\xi^2 = \xi^2_{max}$, respectively. An integer array `numbc(i,k)` should be set to the number of boundary conditions specified (0 for no conditions).

In many problems, physical quantities such as $\tilde{Q}^1$=$\tilde{\rho}$, $\tilde{Q}^2$=$\tilde{T}$ and contravariant velocities $\tilde{u}^{\xi^j}$ (not f^j), are readily known at the boundaries. Then appropriate transformations may be used in the boundary conditions. When Neuman-type boundary conditions or mixed Neuman-Dirichelet boundary conditions on $\tilde{Q}^j$, $j = 1, 2$ are known at the boundaries, derivative of $\tilde{\boldsymbol{Q}}$ with respect to ξ^2 may be related to that of $\boldsymbol{f}$ as follows.

$$\frac{\partial \tilde{\boldsymbol{Q}}}{\partial \xi^2} = \boldsymbol{M}\,\frac{\partial \boldsymbol{f}}{\partial \xi^2} + \frac{\partial \boldsymbol{M}}{\partial \xi^2}\boldsymbol{f}$$

$\partial M/\partial\xi^2$ can be evaluated from the stored global arrays `d1xvra(n,k,j,i)` and `dxdca(n,2,k)` by using the chain rule.

$$\frac{\partial M_j^i}{\partial \xi^2} = \frac{\partial M_j^i}{\partial x^k}\frac{\partial x^k}{\partial \xi^2}$$

Similarly, boundary conditions on contravariant velocities $\tilde{u}^{\xi^m}$ may be related to those of f^j as follows:

$$\begin{aligned}\tilde{u}^{\xi^m} &= \frac{\partial \xi^m}{\partial x^i}\tilde{u}^i \\ &= \frac{\partial \xi^m}{\partial x^i} M_j^{i+2} f^j \;; i = 1,\ldots,3;\; m = 1,\ldots,3\end{aligned}$$

For example, one may specify the adiabatic boundary condition on $\tilde{T}$ $(i = 2)$, i.e., $\partial\tilde{T}/\partial\xi^2 = 0$ at the wall $(k = 0)$ as follow:

$$\left.\begin{array}{lcl} \texttt{bvec}(2,1,1,0) & = & \texttt{vrma}(0,1,2) \\ \texttt{bvec}(2,0,1,0) & = & \texttt{d1cvrm}(2,1,2) \\ \texttt{bvrhs}(2,1) & = & 0 \end{array}\right\} \cdots;\; l = 1,\ldots,nvar$$

$$\texttt{numbc}(2,0) = 1$$

Here, `d1cvrm(2,1,2)` is the local array for $\partial M_l^2/\partial\xi^2$ evaluated at the collocation point $n = 0$.

The following general conditions are applied in `boundary.ins` files provided with the built-in sample applications. At the wall, disturbance temperature $\tilde{Q}^2$ and velocities are set to zero. At farfield, ξ^2_{max}, disturbance temperature $\tilde{Q}^2$, velocities in ξ^1 and ξ^3 directions are set to zero. Then ξ^2-gradient of disturbance velocity in ξ^2 direction is determined from the continuity equation assuming that density $\tilde{Q}^1$ and gradients of density are zero implicitly. [1] No explicit boundary condition on density is required at either end of the boundaries. For the current version of the code, asymptotic conditions (Appendix B) are implemented only for blunt-cone calculations using orthogonal body-intrinsic coordinates. Boundary conditions for the *stationary* $(\omega = 0)$ modes may be different from those of oscillatory disturbances. For example, when the basic laminar flow satisfies the adiabatic wall boundary condition, normal gradient of $\tilde{T}$ for stationary modes may be set to zero (see Mack 1984).

2.2.7 postpr.ins for PSH

This insert file is for postprocessing the data at specified stations to obtain specific output quantities. This file is included in subroutine `wmodes` before any executable

[1] The user may choose to set this velocity component to zero for oscillatory modes with little or no change in the solution.

statement. Hence the user may declare new variables at the top of this insert file, as required. At each streamwise station, the user has access to the following arrays:

`func(n,m,iv,it,iz,lt)` $= \partial^m f^{iv}/\partial\xi^{2,m}$, *iv-th* variable of the *shape function* $\boldsymbol{f}$ for mode (*it,iz*) and its m-th derivatives with respect to ξ^2 at n-th collocation point and ($lt = lprev$) previous step or ($lt = lcurr$) current step (`complex`).
$n = 0, ..., jx$; $m = 0, ..., nx$; $iv = 1, ..., nvar$.

`expr(it,iz,lt)` $= exp^{-\alpha_i \xi^1}$, amplitude modulation owing to exponential growth for mode (*it,iz*) at ($lt = lprev$) or ($lt = lcurr$) step, (`complex`).

`parm(k,it,iz,lt)`, parameters $P_k, k = 1, ..., npar$, such as wavenumbers and frequency (as defined in `definitions`) for mode (it, iz) at the current or previous step (`real`).

The user may also use the available basic-state arrays described in `btrans.ins`, metric arrays described in `metric.ins`, and variable transformation arrays described in `boundary.ins`.

2.2.8 plotps.ins for PSH

This insert file is included in subroutine `wmodes` after `postpr.psh`. It opens two output files for a mode (*it,iz*), as this mode is introduced.

1. Profile data file `pt(it)zp(iz).dat`
2. Station data file `st(it)zp(iz).dat`,

where `it` and `iz` in parentheses are to be substituted by indices of the mode. The user is encouraged to use the same logical statements in naming these output files, although other contents of these files also can be changed.

The Profiles of eigenfunctions and growth rates are written to `pt(it)zp(iz).dat` file at every `npprt` step (station). The value of `npprt` is specified as a constant in the `Definitions` file.

Station (current step) data are printed out to `st(it)zp(iz).dat` every step in ξ^1. It may include parameters, maximum amplitude of different disturbance variables, growth rates along constant ξ^2, amplitudes of massflow and total temperature fluctuations at the location where the massflow for the reference mode (*itr,izr*) is maximum, and their corresponding growth rates, etc.

Some of the station data are also printed out in `0.summary`, the standard output file for PSH. Insert files `postpr.psh` and `plot.psh` are provided so that the user may customize the output for a particular problem.

2.3 Preparing the Definitions

The file `Definitions` is read in a format similar to the free format in Fortran 77. Exceptions are: all items must be separated by spaces (no commas are allowed); the items may be character strings; and the data cannot be continued on the next line. The slash / terminates the reading of data from the line. Empty lines are ignored and can be freely used to enhance readability.

The file `Definitions` consists of various groups of data described in the following:

(1) A single line of text specifying the `problem`.

(2) One line specifying the integer

`jx` the highest index of the collocation points, $j = 0, \cdots, jx$. The number $jx + 1$ of points can reach from as few as 30, say, to as many as 200 or so.

(3) One line specifying the integer

`ncon` the number of constants or zero (if there are no constants).

(4) One line specifying the integer

`npar` the number of parameters. Note that complex parameters count as two real parameters.

(5) One line specifying the type of collocation points for stability analysis, `idpts`:

1 for gauss points in $(-1, 1)$ for spectral version only.

0 for using the basic-state grid points

-1 for stretched grid points

Normally, uniformly spaced grid points are suitable for Hermitian finite difference method used. For high-speed boundary layers, the grid resolution is required not only near the wall but also near the boundary layer edge where the critical layer is located.

(6) One line specifying the stretching in $\xi^2(j)$ with the following five data:

`idtrf` the type of transformation (integer)

`strfp(1)` the minimum value of ξ^2, a, (real)

`strfp(2)` the maximum value of ξ^2, b, (real)

`strfp(3)` the third stretching parameter, or zero (real)

`strfp(4)` the fourth stretching parameter, or zero (real)

The following transformations are available by specifying `idtrf`:

1 Equally spaced grid $\xi^2(j) = a + (b - a) * j/jx$

2 Exponential stretching $\xi^2(j) = a + (b-a) * (r^j - 1)/(r^{jx} - 1)$ where `strfp(3)`$=r^{(jx-1)}$ is the ratio of maximum grid spacing at $\xi^2_m ax$ minimum spacing at $\xi^2_m in$. `strfp(3)` must be greater than or equal to 1.

3 Exponential stretching

$$\xi^2(j) = a + (b-a) * \frac{strfp(3) + 1 - e * \{strfp(3) - 1\}}{e+1}$$

$$e = \{\frac{strfp(3)+1}{strfp(3)-1}\}^{j/jx}$$

4 Logarithmic stretching of

$$\xi^2(j) = a + (b-a) * (ru/rl)$$

with

$$ru = log(strfp(3) + j/jx)/(strfp(3) - j/jx)$$

$$rl = log(strfp(3) + 1)/(strfp(3) - 1)$$

Typically, `strfp(3)` ≈ 1.1.

5 The hyperbolic sine mapping

$$\xi^2(j) = a + \{strfp(3) - a\} \frac{1 + sinh(strfp(4))\frac{j}{jx-c}}{sinh(strfp(4) * c)}$$

where

$$c = \frac{1}{2} strfp(4) log \frac{1 + (e^{strfp(4)} - 1)^{\frac{strfp(3)-a}{b-a}}}{1 + (e^{-strfp(4)} - 1)^{\frac{strfp(3)-a}{b-a}}}$$

$$strfp(3) \geq a, \; strfp(4) \geq 0$$

(7) `nvar` lines, where each line specifies one variable by the following data:

`vname` the name of the disturbance variable f^j, up to 8 characters.

`negv` the ξ^3-symmetry of this variable, where

- 0 indicates no symmetry condition on this variable
- 1 indicates this variable is symmetric
- -1 indicates this variable is antisymmetric

`idsy` the ξ^2-symmetry of this variable, where

0 indicates no symmetry condition on this variable

1 indicates this variable is symmetric

-1 indicates this variable is antisymmetric

(8) `ncon` lines, where each line specifies one constant by the following data:

`cname` the name of the constant C_i, up to 8 characters.

`con` the value of the constant C_i (real)

This section may be missing if `ncon=0`.

(9) `npar` lines, where each line specifies one parameter by the following data:

`pname` the name of the parameter P_k, up to 8 characters.

`ipar` the appearance of the parameter in the problem, where

0 indicates that the parameter appears linear in the stability equations and does not affect the basic state

1 indicates that the parameter appears nonlinear in the stability equations and does not affect the basic state

2 indicates that the parameter affects the basic state

`ityp` the real or complex nature of the parameter, where

0 indicates that the parameter is real

1 indicates that the parameter is the real part of a complex quantity

2 indicates that the parameter is the imaginary part of a complex quantity

Real and imaginary parts of complex quantities must immediately follow each other.

Although tedious to describe, the `Definitions` are usually quick done. However, it is important to prepare this file carefully since the sequence of data must be correct.

Chapter 3

Tasks and Parameters for LSH

The file `Parameters` contains the sequence of tasks for the code LSH to perform. The file `Parameters` is read in a format similar to the free format in Fortran 77. Exceptions are: all items must be separated by spaces (no commas are allowed); the items may be character strings; and the data cannot be continued on the next line. The slash / terminates the reading of data from the line. Empty lines are ignored and can be freely used to enhance readability.
The code can perform the following tasks:

`itask` = 1	**local**	procedure to find a single eigenvalue
`itask` = 2	**local**	procedure to find eigenvalue and eigenfunction
`itask` = 3	**table**	of eigenvalues in 1, 2, or 3 dimensions
`itask` = 4	**curve**	of solutions in the plane of two parameters
`itask` = -1	close and reopen the file `Parameters` after pause	
`itask` = -2	execute a system command	
`itask` =-999	terminate	

These tasks and their required input are described in the following sections.

3.1 Single Eigenvalues

Procedure to find spectra of eigenvalues is not yet implemented in the Hermitian version of LSH. In contrast to the global procedure to find spectra, the local procedure searches for the value of one real parameter (for real problems) or the values of two real parameters (for complex problems) that solve the characteristic equation. For complex problems, the two real parameters are not necessarily the real and imaginary part of a complex quantity. We denote these two parameters as the "first and second eigenvalue parameter." To obtain an eigenvalue with the local procedure, an initial guess must be specified. If such an estimate is not available, `itask=0` can help to provide many eigenvalues (if the eigenvalue appears linearly in the stability equations).

Otherwise, `itask=3` (table) can be used to analytically continue an eigenvalue known at different parameters to the desired point.
To obtain a single eigenvalue for a certain parameter combination, the file *Parameters* requires the following information:

(a) One line with `itask`:
1	to request an eigenvalue

(b) One line with the following data:
`iev1`	the index of the first eigenvalue parameter
`iev2`	the index of the second eigenvalue parameter (for complex problems only)
-1	an optional specification to reset the eigenfunction

(c) `npar` lines, where each line specifies:
`par(k)`	the value of parameter P_k. The values of `par(iev1)` and fortpar(iev2) are set to the initial guess.

For the first eigenvalue, the iteration uses the initial guess for the eigenvalue and a default setting for the eigenvector. For subsequent eigenvalues, the converged eigenvalue and eigenvector of the previous solution is used as an initial guess to speed up the convergence. The optional reset of the eigenvector to the default setting breaks this routine. This option is provided to avoid problems with changes from one eigenmode to another, e.g., from an antisymmetric to a symmetric mode. This option should be used if the mode changes or if the parameters change drastically.

3.2 Eigenvalue and Eigenfunction

This task is identical with `itask=1` except the coefficients of the Chebyshev series and the values of the variables at the collocation points are evaluated. Except for itask, the specifications are the same as in the previous section:

(a) One line with `itask`:
2	to request an eigenvalue

(b) One line with the following data:
`iev1`	the index of the first eigenvalue parameter
`iev2`	the index of the second eigenvalue parameter (for complex problems only)
-1	an optional specification to reset the eigenfunction

(c) `npar` lines, where each line specifies:
`par(k)`	the value of parameter P_k. The values of `par(iev1)` and `par(iev2)` are set to the initial guess.

The real and imaginary parts of eigenfunction are stored in file `0.function`. This file may be use by PSH as initial data. The eigenvector is normalized to a reasonable scale but determined only within a complex factor. Amplitude and phase of eigen function are stored in `0.fampha` for ploting purposes.

3.3 Tables of Eigenvalues

This task enables calculation of one-, two-, or three-dimensional tables of eigenvalues. The local procedure (`itask=1,2`) must be performed at least once to provide a starting point for this task. The values of `iev1` and `iev2` that specify the eigenvalue are taken from the previous task. The required input is:

(a) One line with `itask`:
    3 to request a table of eigenvalues coefficients

(b) One line specifying the dimension of the table:
    `ndim` the dimension of the table (integer)

(c) `ndim` lines, where each line specifies:
    `kpar` the index of the parameter to be varied (integer)
    `nstep` the number of steps to be taken (integer)
    `step` the step size (real)

The first parameter specified is varied in the innermost loop. The eigenvalues calculated are stored in file `0.table`.

3.4 Curves in Parameter Planes

This task enables the direct calculation of neutral curves, curves of constant amplification, and other useful curves in the plane of two parameters with a third parameter completing the complex eigenvalue problem. The local procedure (`itask=1,2`) must be performed at least once to provide a starting point for this task.

To describe the input for a complex problem, we imagine a three-dimensional coordinate system with the horizontal, vertical, and normal axes formed by the three parameters `par(iv1)`, `par(iv2)`, and `par(iv3)`, respectively, and a curve in the plane spanned by `par(iv1)` and `par(iv2)`. Beginning at some starting point provided by the local procedure, we want to proceed in one of the two possible directions and find new points of the curve. All but the three parameters used as coordinates maintain the starting values along the curve. In particular, for a neutral curve, the growth rate in the starting point must be zero.

Since the parameters involved may have different orders of magnitude, we have to choose proper scale values. For this step, we envision a plot of the curve on a sheet

of paper and choose the scales according to the rules of nomography so that the final curve would look "nice" without being squeezed in either axis. In the scaled plot, the distance between points should be neither too large nor unnecessarily small. Also, the plot has a "window" defined by the minimum and maximum of the variables plotted.

The input required is as follows:

(a) One line with `itask`:

4	to request a curve in a parameter plane

(b) Two lines specifying data for the "horizontal" and "vertical" direction, respectively:

`ivi,2`	the index of the associated parameter (integer)
`vgrid1,2`	the scale value for this parameter (real)
`vmin1,2`	the minimum value (real)
`vmax1,2`	the maximum value (real)

(c) For complex problems only: one line specifying

`iv3`	the index of the second eigenvalue parameter (integer)

(d) One line specifying direction and step size along the curve in scaled variables:

`angle`	the initial direction in the plane of the curve as the positive (counter-clockwise) or negative angle measured in degrees from the horizontal axis
`radius`	the distance between points along the curve

(e) One line specifying the number of steps:

`npoints`	the number of points, or arc length in multiples of the radius, to be traced along the curve

The eigenvalues calculated are stored in file `0.curve`.

The iteration for the eigensolution can neither use `par(iv1)` (if the curve tangent is horizontal) nor `par(iv2)` (if the tangent is vertical) as the first eigenvalue parameter but internally iterates normal to the curve in the plane of `par(iv1)`, `par(iv2)`. for complex problems, the second eigenvalue parameter is `par(iv3)`.

A safe recipe for the choice of the various data cannot be given here since it depends on the unknown properties of the curve. A reasonable choice for the scale values would be the order of magnitude or the physical values of the parameters `par(iv1)`, `par(iv2)` of some point at the curve. For plane Poiseuille flow, with `par(iv1)=Re`, `par(iv2)`$= \alpha_r$, and `par(iv3)`$= \omega_r$, the choice `vgrid1`$= 10000$, `vgrid2`$= 1$ would be almost the same as `vgrid`$= 20000$, `vgrid2`$= 2$, and be as good as `vgrid1`$= 5772$, `vgrid2`$= 1$. Note, however, that the radius is the distance in scaled variables. With a radius of 0.01, the maximum physical steps in the horizontal direction would be 100, 200, or 57.72 in the three examples, while the maximum vertical steps would be 0.01, 0.02, or 0.01. The steps should be small enough to obtain a smooth plot (and

rapid convergence) and because problems may arise if the distance between multiple branches of the curve is smaller than the radius. Neutral curves can have cusp-like sharp turns, as for the boundary-layer over a rotating disk. If the steps are too large for convergence, or if the specified radius exceeds $\pi/8$th of the curve radius of curvature, the radius will be automatically divided by increasing powers of 2, and hence the code may produce 2, 4, or more points per step. The procedure attempts to double the reduced internal step size after every point until it proceeds with the original radius. Hence, the continuation procedure will slow down in regions of strong curvature. This internal testing consumes additional CPU time, and one may as well run the whole curve with a smaller radius instead.

Ideally, the angle (in degrees) would be the positive or negative angle between the horizontal and the curve tangent in the starting point. The two signs correspond to the two directions in which the procedure can proceed. The choice of the angle is not critical as long as the radius is sufficiently small and the initial direction is not normal to the curve. Trial and error with `npoints=4` and a small radius will help (the angle and radius for the last point are printed). Note that the angle changes with the ratio `vgrid1/vgrid2`.

The procedure terminates if the number of (full) steps exceeds npoints, if the distance between any point (after the third) and starting point is less than the radius (to prevent unnecessary loops through closed curves), and if the curve crosses the window given by the minimum and maximum values. It is often desirable to specify extreme values for npoints or the window size.

3.5 Close and Reopen the Input File

This task enables to suspend the program execution, edit or replace the *Parameters*, and resume execution either on a multiwindow workstation or on a terminal under Unix. The input is as follows:

(a) One line with `itask`:
  −1 to request a pause for editing the `Parameter` file

The program will close the file `Parameters` and pause until continuation is requested. During this time, the file can be changed. After continuation, the new `Parameters` will be read starting at the top. This task is especially helpful in the initial study of a new problem without guidance on proper parameter ranges. The `Parameters` file can be displayed and edited in one window while test runs are performed in another. Note that all input is saved in the input-log file `0.input` for later inspection and reruns.

3.6 Execute System Command

This task enables to execute system commands during execution of the code. The input is as follows:

(a) One line with **itask**:

-2	to request execution of a system command
text	the text passed to the system call

Deviating from the general format, the text may contain spaces and is terminated by a slash / or the end of the line. The command is typically used to copy or move output files into safe files before they are overwritten by one of the consecutive tasks. A typical application is securing part of a curve before the continuation in the opposite direction is requested.

Chapter 4

Control for PSH

The file `Control` is read in a format similar to the free format in Fortran 77. Exceptions are: all items must be separated by spaces (no commas are allowed); the items may be character strings; and the data cannot be continued on the next line. The slash / terminates the reading of data from the line. Empty lines are ignored and can be freely used to enhance readability.

The file `Control` consist of various groups of data described in the following:

(1) A single line of text specifying the `problem`.

(2) One line specifying the following four data

- `nstart` station number at which initial data is specified
- `nfinal` final station number
- `ndelta` station numbers per step
- `step` integration step size in ξ^1

(3) One line specifying the integer

- `inlt` the flag controlling whether the nonlinear terms are implicitly evaluated at the current step or explicitly at the previous step

  - 0 explicit
  - 1 implicit

(4) One line specifying the integer

- `itx` the maximum number of iterations to be performed for each mode per step

(4) One line specifying the following two data

- `tpar` temporal frequency of the fundamental mode

`zpar` wave number in the ξ^3 coordinate direction for the fundamental mode

(6) One line specifying the two indices
- `itr` temporal index of the reference mode
- `izr` index of the wave number in the ξ^3 direction for the reference mode

(7) One line specifying the following two thresholds
- `thron` threshold maximum magnitude of the nonlinear forcing terms for a new mode, above which a new mode is created and integrated
- `throff` threshold amplitude of the `normv` disturbance variable below which an existing *decaying* mode will be discarded

(8) One line specifying the number of mode-control lines `nmgr`.

(9) `nmgr` lines, where each line specifies control of a mode by the following data:
- `it` the temporal frequency index of the mode in multiples of `tpar`.
- `iz` the ξ^3-wavenumber index of the mode in multiples of `zpar`.
- `mgr` mode control parameter where
  - 0 indicates that the mode is calculated by including nonlinear effects of other modes on itself but neglecting its effect on other modes
  - 1 the mode is calculated and its interaction with other modes is included, and the real part of its wavenumbers is determined by the phase-locking condition with the reference mode
  - 2 the mode is calculated and its interaction with other modes is included, and the real part of its wavenumber is determined independently by imposing a restriction on its norm (default)
  - 3 the mode is calculated and the nonlinear effects on this mode is neglected by setting the right hand side to zero. Its wavenumber is determined by imposing a restriction on its norm
  - -1 indicates that the mode is to be neglected altogether

(10) One line specifying the number of initial modes `nmodes`.

(11) `nmodes` lines, where each line specifies an initial mode by the following data:
- `it` the temporal frequency index of the mode

`iz` the ξ^3-wavenumber index of the mode

`mgs` mode control option for the initial mode, as described above in item 7. This value of `mgs` will overwrite `mgr` for this mode if it is specified in item 7.

`amp0` amplitude of the initial mode for the `mvar` variable chosen for imposition of the norm constraint

`phs0` phase of the initial mode for the `normv` variable chosen for imposition of the norm constraint

Not that the indices of the modes specified here should be within the limits specified by parameters (`modt`) and (`modzm, modzp`) specified in `common.ins`. A summary of output data for every step of `psh` run is written out to the file `0.summary`.

Chapter 5

Installation

The code is delivered in *tar* format on a cartridge tape compatible with Silicon Graphics standard cartridge tape drives. To provide full functionality, the code should be installed in the user directory. No *root* privileges are required. Insert the tape cartridge into the drive. To conform with the original, make a new directory *w.psh* and read the *tar* file:

```
$ mkdir w.psh
$ cd w.psh
$ tar xvo    ( or  tar xvof /dev/??? )
$ ls
```

If the default device */dev/tape* is not available on your workstation, use the proper device name for the cartridge tape drive on a remote workstation. After the installation is complete, there will be one file `README`, a *Makefile*, a compressed *tar* file of the code that should be saved, and a series of directories. First, read `README` to get an update on changes, bugs, or other news.
To test for successful installation, execute the following commands:

```
$ cd c.cone
$ ls
$ make
$ lsh
$ ls
```

The comparison of the file lists should show a series of output files such as `0.input`. The names of all output files start with `0.` which simplifies their handling or removal. If the code ran error-free to completion, the output files should be identical with those in the subdirectory *w.results*, except for the date. To verify proper function of the code, check with

```
$ diff .  w.results
```

This command should reveal no differences for `0.*` files. The command

```
$ make clean
```

removes the problem specific object files in *../w.objs*, the executable files **lsh** and **psh**, and prompts whether you want to delete the output files in the current directory. New applications must be prepared in directories on the same level, i.e., in directories *w.psh/c.new-appl* to be compatible with the various Makefiles.

Chapter 6

Examples of Applications

6.1 Hypersonic Flow over a Blunt Cone

Insert files and input files for studying stability of hypersonic flow over a blunt cone are set up in directory *c.bluntc1*. These files are used as examples in this section to illustrate how the user may build his own physics insert files and input files to tackle a new problem.

6.1.1 metric.ins

Metric terms for transformation from Cartesian coordinates (x, y, z) to axisymmetric body-intrinsic coordinates ($\xi^1 = s$, $\xi^2 = \eta$, $\xi^3 = \phi$) and local Cartesian coordinates aligned with the cone surface are given here. They are coded into the *metric.ins* insert file.

Body-Intrinsic Coordinates

This coordinate system (Figure 6.1) is used when `icoord=1` is specified in `Definitions`. Transformation is from Cartesian coordinates:

$$x^1 = x$$
$$x^2 = y$$
$$x^3 = z$$

to:

ξ^1 = s; arc-length along the body
ξ^2 = η; normal distance from the body surface
ξ^3 = ϕ; azimuthal angle

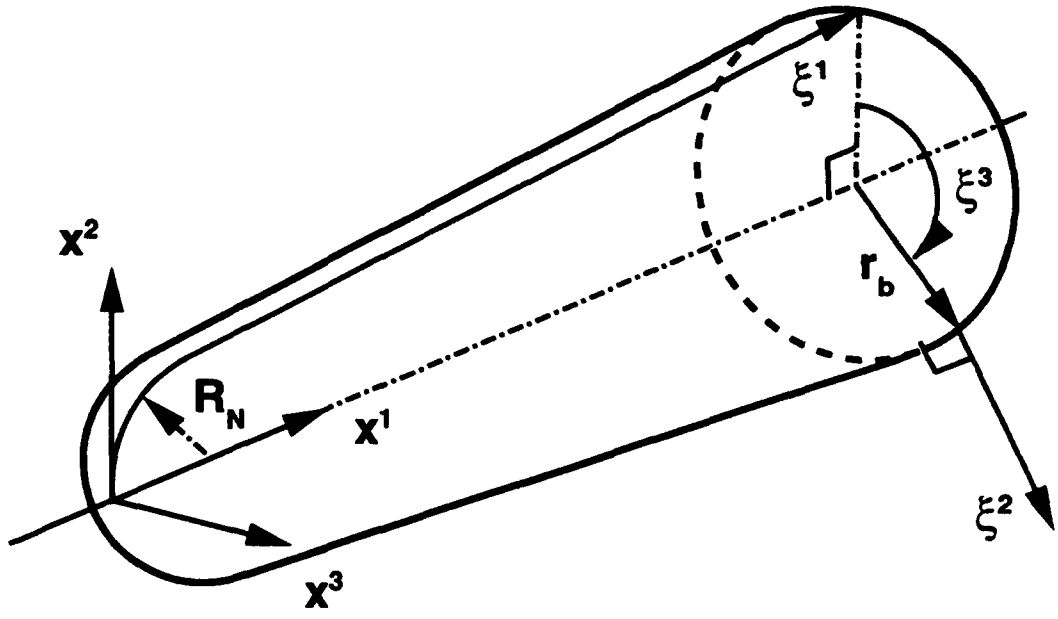

Figure 6.1: Body-Intrinsic Coordinate System Used for Stability Analysis

$$
\begin{aligned}
x &= \{x_b(s) - \eta sin(\theta_c(s))\} \\
y &= \{r_b(s) + \eta cos(\theta_c(s))\}\, sin(\phi) \\
z &= \{r_b(s) + \eta cos(\theta_c(s))\}\, cos(\phi)
\end{aligned}
$$

where $r_b(s)$ is the radius of the body, $x_b(s)$ is the axial distance, and $tan\{\theta_c(s)\}$ is the slope, at the body surface ($\eta = 0$).

First derivatives are :

$$
\begin{aligned}
\frac{\partial x^1}{\partial \xi^1} &= \frac{\partial x}{\partial s} \\
&= \frac{dx_b}{ds} - \eta\, cos(\theta_c(s))\, \frac{d\theta_c}{ds} \\
\frac{\partial x^1}{\partial \xi^2} &= \frac{\partial x}{\partial \eta} \\
&= -\, sin(\theta_c(s)) \\
\frac{\partial x^1}{\partial \xi^3} &= \frac{\partial x}{\partial \phi} \\
&= 0
\end{aligned}
$$

$$
\begin{aligned}
\frac{\partial x^2}{\partial \xi^1} &= \frac{\partial y}{\partial s} \\
&= \{\frac{dr_b}{ds} - \eta\, sin(\theta_c(s))\, \frac{d\theta_c}{ds}\}\, sin(\phi) \\
\frac{\partial x^2}{\partial \xi^2} &= \frac{\partial y}{\partial \eta} \\
&= cos(\theta_c(s))\, sin(\phi) \\
\frac{\partial x^2}{\partial \xi^3} &= \frac{\partial y}{\partial \phi} \\
&= \{r_b + \eta cos(\theta_c(s))\}\, cos(\phi)
\end{aligned}
$$

$$
\begin{aligned}
\frac{\partial x^3}{\partial \xi^1} &= \frac{\partial z}{\partial s} \\
&= \{\frac{dr_b}{ds} - \eta\, sin(\theta_c(s))\, \frac{d\theta_c}{ds}\}\, cos(\phi) \\
\frac{\partial x^3}{\partial \xi^2} &= \frac{\partial z}{\partial \eta} \\
&= cos(\theta_c(s))\, cos(\phi) \\
\frac{\partial x^3}{\partial \xi^3} &= \frac{\partial z}{\partial \phi} \\
&= -\{r_b + \eta cos(\theta_c(s))\}\, sin(\phi)
\end{aligned}
$$

Second derivatives are:

$$
\begin{aligned}
\frac{\partial^2 x^1}{\partial \xi^1 \partial \xi^1} &= \frac{\partial^2 x}{\partial s^2} \\
&= \frac{d^2 x_b}{ds^2} + \eta\, sin(\theta_c(s)) \left(\frac{d\theta_c}{ds}\right)^2 - \eta\, cos(\theta_c(s))\, \frac{d^2\theta_c}{ds^2} \\
\frac{\partial^2 x^1}{\partial \xi^1 \partial \xi^2} &= \frac{\partial^2 x}{\partial s \partial \eta} \\
&= -cos(\theta_c(s))\, \frac{d\theta_c}{ds} \\
\frac{\partial^2 x^1}{\partial \xi^1 \partial \xi^3} &= \frac{\partial^2 x}{\partial s \partial \phi} \\
&= 0
\end{aligned}
$$

$$
\begin{aligned}
\frac{\partial^2 x^2}{\partial \xi^1 \partial \xi^1} &= \frac{\partial^2 y}{\partial s^2} \\
&= \{\frac{d^2 r_b}{ds^2} - \eta\, cos(\theta_c(s)) \left(\frac{d\theta_c}{ds}\right)^2 - \eta\, sin(\theta_c(s))\, \frac{d^2\theta_c}{ds^2}\} sin\, \phi
\end{aligned}
$$

$$
\begin{aligned}
\frac{\partial^2 x^2}{\partial \xi^1 \partial \xi^2} &= \frac{\partial^2 y}{\partial s \partial \eta} \\
&= -\sin(\theta_c(s)) \frac{d\theta_c}{ds} \sin\phi \\
\frac{\partial^2 x^2}{\partial \xi^1 \partial \xi^3} &= \frac{\partial^2 y}{\partial s \partial \phi} \\
&= \{\frac{dr_b}{ds} - \eta \sin(\theta_c(s)) \frac{d\theta_c}{ds}\} \cos(\phi) \\
\frac{\partial^2 x^2}{\partial \xi^2 \partial \xi^3} &= \frac{\partial^2 y}{\partial \eta \partial \phi} \\
&= \cos(\theta_c(s)) \cos\phi \\
\frac{\partial^2 x^2}{\partial \xi^3 \partial \xi^3} &= \frac{\partial^2 y}{\partial \phi \partial \phi} \\
&= -\{r_b + \eta \cos(\theta_c(s))\} \sin(\phi)
\end{aligned}
$$

$$
\begin{aligned}
\frac{\partial^2 x^3}{\partial \xi^1 \partial \xi^1} &= \frac{\partial^2 z}{\partial s^2} \\
&= \{\frac{d^2 r_b}{ds^2} - \eta \cos(\theta_c(s)) \left(\frac{d\theta_c}{ds}\right)^2 - \eta \sin(\theta_c(s)) \frac{d^2\theta_c}{ds^2}\} \cos\phi \\
\frac{\partial^2 x^3}{\partial \xi^1 \partial \xi^2} &= \frac{\partial^2 z}{\partial s \partial \eta} \\
&= -\sin(\theta_c(s)) \frac{d\theta_c}{ds} \cos\phi \\
\frac{\partial^2 x^3}{\partial \xi^1 \partial \xi^3} &= \frac{\partial^2 z}{\partial s \partial \phi} \\
&= -\{\frac{dr_b}{ds} - \eta \sin(\theta_c(s)) \frac{d\theta_c}{ds}\} \sin(\phi) \\
\frac{\partial^2 x^3}{\partial \xi^2 \partial \xi^3} &= \frac{\partial^2 z}{\partial \eta \partial \phi} \\
&= -\cos(\theta_c(s)) \sin\phi \\
\frac{\partial^2 x^3}{\partial \xi^3 \partial \xi^3} &= \frac{\partial^2 z}{\partial \phi \partial \phi} \\
&= -\{r_b + \eta \cos(\theta_c(s))\} \cos(\phi)
\end{aligned}
$$

For the straight cone region, with uniform grid distribution in s, there are only two nonzero third derivative terms.

$$
\begin{aligned}
\frac{\partial^3 x^2}{\partial \xi^1 \partial \xi^3 \partial \xi^3} &= \frac{\partial^2 y}{\partial s \partial \phi \partial \phi} \\
&= -\{\frac{dr_b}{ds} - \eta \sin(\theta_c(s)) \frac{d\theta_c}{ds}\} \sin(\phi)
\end{aligned}
$$

$$\begin{aligned}\frac{\partial^3 x^2}{\partial\xi^2\partial\xi^3\partial\xi^3} &= \frac{\partial^2 y}{\partial\eta\partial\phi\partial\phi} \\ &= -cos(\theta_c(s))\,sin\,\phi\end{aligned}$$

Local Cartesian Coordinates

This coordinate system is used for stability analysis when `icoord=0` is specified. Here, all curvature terms and cone divergence are neglected.

$$\begin{aligned} x &= s \\ y &= \eta \\ z &= z \end{aligned}$$

$$\begin{aligned}\frac{\partial x^1}{\partial\xi^1} &= \frac{\partial x}{\partial s} = 1 \\ \frac{\partial x^2}{\partial\xi^2} &= \frac{\partial y}{\partial\eta} = 1 \\ \frac{\partial x^3}{\partial\xi^3} &= \frac{\partial z}{\partial z} = 1\end{aligned}$$

Since the basic flow is axisymmetric, and disturbances are periodic in ξ^3 direction with wave number n_ϕ, ϕ is set to zero. with $\phi = 0$ in the above metric terms.

6.1.2 gridbs.ins

Subroutine `gridbs` assigns the basic state grid simply by reading it in along with the basic state solution. Auxiliary routines that `gridbs` calls to accomplish this are described more fully in Appendix C. Of these, three are specific to the basic state flow solver used. They are: `bsiois`, `bpstda`, and `scale` .

Subroutines `bsiois` and `bpstda` read in the station data and flow profiles, respectively, from a FORTRAN formatted file `bpse.inp`. This data file contains the basic state solution and derivatives generated by the Thin Layer Navier-Stokes (TNLS) code (Esfahanian 1991). The TNLS code uses a *nonorthogonal* axisymmetric body-intrinsic grid. The equations are solved in the computational domain (I,J). The outer-most grid line in the streamwise direction $\xi^1(JMAX)$ is the bow shock. Grid lines in the streamwise direction (J+constant) are generated by dividing the distance between the shock and the cone surface by a stretching function (stored in an array `an(J)`). Grid lines (I=constant) are normal to the cone surface.

bsiois

Subroutine `bsiois` reads in station data and profile data and stores the station data in the array `stndat`. Profile data are read from the sequential data file for error checking. It also sets the scales for the basic state solution by calling the subroutine `therm` (see appendix). The relevant `open`, `read` and station data assignment statements are given below.

```
      i = inpprf
      open(unit=i,file=profil,access='sequential',form='formatted',
     |     status='old')
      rewind(i)


c  read in all the station data (except the profiles themse. ves) from
c  input and store them ... (assume that station number nstn+1
c  follows station number nstn in the input file.)

      lcp=len(charv)
      do 10 iv=1,nc
       do 10 k=1,lc
        write(charv(iv)(k:k),'(a1)') ' '
   10 continue

      read(i,'(a80)') charv(1)

c tcone  = cone half angle (degrees)
c scone  = arc length along surface at end of spherical nose (m)
c aminf  = free-stream mach number
c reinf  = free-stream reynolds number
c tinf   = free-stream static temperature (k)
c pinf   = free-stream static pressure (psia)
c reflen = spherical nose radius (m) (reference length)
c betaa  = basic state grid stretching parameter
c tw     = basic state grid stretching parameter
c pr     = prandtl number
      read(i,*) tcone,scone,aminf,reinf,tinf,pinf,reflen,
     |          betaa,tw,pr

c  specify the free-stream mach number, reference length, and inverse
c  reynolds number by making the 1's digit in mdlair equal to 1 and the
c  10's digit equal to 1 ...

      mdlair = iabs(nint(con(2)))
      mdlair = mdlair - mod(mdlair,100) + 11
```

```
c  compute the ratio of specific heats (gamma), reference velocity, ...
c  set the local pressure equal to 1 and disregard the equation of state
c  data here.

      fsrei = one/reinf
      call therm (mdlair,0,tinf,fspres,bsrrho,bsrvel,reflen,dypdep,
     |            aminf,fsrei,pr,gae,cpe,dyvisc,sve,coe,one,one,r0s,
     |            r1s,r2s,cp0,cp1,cp2,cp3,cp4,fv0,fv1,fv2,fv3,fv4,
     |            sv0,sv1,sv2,sv3,sv4,co0,co1,co2,co3,co4)


c  vahid's velocity scale is sqrt(r*tinf), where r = gas constant.

      bsrvel = bsrvel/sqrt(gae)/aminf

      realv( 1) = reflen
      realv( 2) = reflen

c  xi^{3} = circumferential angle (dimensionless) ...

      realv( 3) = one
      realv( 4) = reflen/bsrvel

      realv( 5) = bsrrho
      realv( 6) = tinf
      realv( 7) = bsrvel
      realv( 8) = bsrvel

      realv( 9) = pinf
      realv(10) = gae
      realv(11) = aminf
      realv(12) = reinf
      realv(13) = pr
      realv(14) = tcone
      realv(15) = scone
      realv(16) = betaa
      realv(17) = tw

      j = 0
 17   j = j +1
      if(j+1.le.jmax) j=j+1
      read(i,902,iostat=ie(1))  an(j),dadc2(j),d2adc2(j),d3adc2(j)
      go to 17
  902 format(1x,4e23.16)
```

```
c  store the stretching function and its derivatives in realv ...

      nr0=18
      do 20 j=0,ja
         realv(nr0+j)        = an(j)
         realv(nr0+j+ja+1)   = dadc2(j)
         realv(nr0+j+2*ja+1) = d2adc2(j)
         realv(nr0+j+3*ja+1) = d3adc2(j)
  20  continue

      nrp=17+4*(ja+1)


c  read to the next station while checking for errors in the input file
      istn   = 0
      nsloc  = 1
   50 continue
          read(i,901,iostat=ie(1),end=900)
     |         nstn,np,arclen,

c dependent variables at shock

     |         qs,rhos,ts,

c grid data at body

     |         xb,yb,xbdc1,ybdc1,xbdc2,ybdc2,xbdc3,ybdc3,

c grid data at shock

     |         xs,ys,xsdc1,ysdc1,xsdc2,ysdc2,xsdc3,ysdc3


         do 55 n=0,np
          read(i,911,iostat=ie(1)) nread,xij,yij,
     |                             bsoln(1),dbdc(2,1),d2bdc2(2,2,1),
     |                             bsoln(2),dbdc(2,2),d2bdc2(2,2,2),
     |                             bsoln(3),dbdc(2,3),d2bdc2(2,2,3),
     |                             bsoln(4),dbdc(2,4),d2bdc2(2,2,4)
55       continue

c  if the entire dataset has been read successfully, increment the loop
c  counter and store the station data ...

         istn=istn+1
```

```
c         shock standoff distance

          sns = dsqrt((xs-xb)**2 + (ys -yb)**2)

          stndat( 0,istn) = nsloc
          stndat( 1,istn) = arclen
          stndat( 2,istn) = np+1
          stndat( 3,istn) = 0          ! xi^{3}
          stndat( 4,istn) = reflen
          stndat( 5,istn) = qs
          stndat( 6,istn) = rhos
          stndat( 7,istn) = ts
          stndat( 8,istn) = sns
          stndat( 9,istn) = xb
          stndat(10,istn) = yb
          stndat(11,istn) = xs
          stndat(12,istn) = ys
          stndat(13,istn) = xbdc1
          stndat(14,istn) = ybdc1
          stndat(15,istn) = xbdc2
          stndat(16,istn) = ybdc2
          stndat(17,istn) = xbdc3
          stndat(18,istn) = ybdc3

          stndat(19,istn) = xsdc1
          stndat(20,istn) = ysdc1
          stndat(21,istn) = xsdc2
          stndat(22,istn) = ysdc2
          stndat(23,istn) = xsdc3
          stndat(24,istn) = ysdc3

c         end if

          nsloc  = nsloc + (np+1)
         go to 50

  900 continue

c  close the input file ...

       close(i)


  901     format(2i5,20e23.15)
```

```
 911     format(i10,14e23.15)
```

bpstda

This subroutine opens the sequential data set named `profil`, connects it to unit `inpprf`, and reads the basic state profiles. It transforms the Cartesian velocity components calculated by the TNLS code to physical contravariant components in the stability analysis coordinates. Then it writes out these profiles to the direct access file named `daprof` which has already been opened and connected to unit `iscr` by `bdiois`. The relevant `open` and `read/write` statements are given below.

```
      logical*(1logic) logicv(0:*)
      character charv(0:*)*(*)
      integer*(lint) intv(-6:*)
      real*(lreal) stndat(0:nstapr,mxnsta),realv(0:*)
      dimension ie(*)

c  nts  = number of thermodynamic state variables in equation of state
c  nbsv = number of basic state variables.
c  j2   = number of basic state grid points.

      parameter (one=1,zro=0,nts=2,nbsv=4,j2=199)
      real*(lreal) an,z,bsoln(nbsv),dbdc(4,nbsv),d2bdc2(4,4,nbsv)

      character profil*(*),daprof*(*)

      i = inpprf
      open(unit=i,file=profil,access='sequential',form='formatted',
     |     status='old')
      rewind(i)

      read(i,'(a80)') charv(1)
      read(i,*) tcone,scone,aminf,reinf,tinf,pinf,reflen,
     |          betaa,tw,pr

c  wall normal stretching function and its derivatives ....

      j =0
      ja=0
      read(i,902,iostat=ie(1))  an
  902 format(1x,4e23.16)
      anl=an
```

```
   17 if (ie(1).eq.0) then
         write(6,902) an
         if (abs(one-an).le.1.e-3*abs(an-anl)) go to 18
      else
         write(6,'(//,1h ,72(1h*),//,3x,''subroutine bpstda: error'',
     |         '' occurred while reading point j='',i3,/,3x,''of '',
     |         ''stretching function.  stop!'',//,1h ,72(1h*),//)') j
         stop
      end if
      anl=an
      ja=ja+1
      if (j+1.le.j2) j=j+1
      read(i,902,iostat=ie(1))  an
      go to 17

   18 if (j.ne.ja) then
         write(6,'(//,1h ,72(1h*),//,3x,''subroutine bpstda: increase'',
     |           '' dimension j2 to at least '',i3,''.  stop!'',//,
     |           1h ,72(1h*))') ja
         stop
      end if

c  number of stations in file ...

      read(i,909,iostat=ie(1)) nstmax
  909 format(i10)
      if (ie(1).ne.0) then
         write(6,'(//,1h ,72(1h*),//,3x,''subroutine bpstda: error '',
     |        ''occurred while reading total number'',/,3x,''of '',
     |        ''stations.  stop!'',//,1h ,72(1h*),//)')
         stop
      end if

c  read in all the station data (except the profiles themselves) from
c  input and store them ... (assume that station number nstn+1
c  follows station number nstn in the input file.)

      pi     = 4*datan(1.d0)
      thetac = tcone*pi/180.d0

      gamma = realv(10)
      aminf = realv(11)

      do 300 k=1,intv(2)
```

```
      read(i,'(2i5)') nstn,np
      key  = nint(stndat(0,k))
      do 85 n=0,np

       do 80 i3=1,nbsv
        bsoln(i3) = zro
        do 75 i2=1,4
         dbdc(i2,i3) = zro
         do 70 i1=1,4
          d2bdc2(i1,i2,i3) = zro
 70      continue
 75     continue
 80    continue

       read(i,911,iostat=ie(1)) nread,xij,yij,
     |                          bsoln(1),dbdc(2,1),d2bdc2(2,2,1),
     |                          bsoln(2),dbdc(2,2),d2bdc2(2,2,2),
     |                          bsoln(3),dbdc(2,3),d2bdc2(2,2,3),
     |                          bsoln(4),dbdc(2,4),d2bdc2(2,2,4)
911   format(i10,14e23.15)

       if (ie(1).eq.0) then

c  assume for now that the angle thetac between the body axis and the
c  surface is a constant.

         if (n.eq.0) rb=yij
         z = (yij-rb)/cos(thetac)

c  transform the velocity components from axial and radial to
c  tangential and normal.

         utan =  bsoln(3)*dcos(thetac) + bsoln(4)*dsin(thetac)
         unor = -bsoln(3)*dsin(thetac) + bsoln(4)*dcos(thetac)

         bsoln(3) = utan
         bsoln(4) = unor

         write(iscr,rec=key+n) z,(bsoln(i3),i3=1,nbsv)
       else

c  shouldn't be any errors because subroutine bsiois already
c  successfully read in file.

         write(6,*) 'stop: error in bpstda.'
```

```
          stop
        end if
        nstnl = nstn

   85 continue

300    continue

       return
       end
```

6.1.3 bstate.ins

Subroutine `bstate` reads in and interpolates the basic state profiles from the direct access file created by subroutine `gridbs`. The method by which it does this is independent of the input and output routines required to read in the basic state, and so need not be changed for different basic state flow solvers. It is described more fully in Appendix C.

6.1.4 btrans.ins

This insert file is written to transform basic state velocity components specified in *bstate.ins* into either Cartesian or physical contravariant velocity components (of the stability coordinate), and their derivatives into derivatives with respect to stability coordinates. As physical contravariant velocity components of the basic flow are defined here, $IBVEL = 1$ is specified. If the basic flow is assumed to be parallel (`mcoord=0`) then normal (to the surface) component of velocity and its derivatives with respect to η are set to zero. All the streamwise derivatives (with respect to s) are also set to zero. If `mcoord=1` the aforementioned nonparallel terms of the basic state are included in the analysis.

Also defined in this insert file are the equation of state for a perfect gas

$$p = \frac{\rho T}{\gamma M_\infty^2}$$

and specific enthalpy

$$h = C_p T$$

and their derivatives with respect to thermodynamic variables. Specific heat at constant pressure C_p, viscosities μ_1 and μ_2, and thermal conductivity κ of the basic state, and their derivatives with respect to temperature are specified by calling `therme` entry statement in subroutine `therm` (see Appendix B). Constant Prandtl number of 0.72, specific heat ratio $\gamma = 1.4$, and the Sutherland Law for viscosities are used.

6.1.5 boundary.ins

The present insert file is set up so that the following sets of conditions may be chosen by the user by specifying the array `idbc(nvar)` in the input file `Definitions`.

1. homogeneous conditions at both boundaries for both stationary and oscillatory modes; $idbc = 1$.

2. homogeneous conditions at both boundaries for both stationary and oscillatory modes except temperature gradient of stationary modes are set to zero at the wall; $idbc = 2$.

3. homogeneous conditions at the wall for both stationary and oscillatory modes and asymptotic boundary conditions at the farfield; $idbc = 3$.

4. homogeneous conditions for both stationary and oscillatory modes at the wall except for normal temperature gradient and asymptotic boundary conditions at the farfield; $idbc = 4$.

If homogeneous conditions are chosen, the outer boundary should be located at a larger distance from the wall than that used with asymptotic conditions.

6.1.6 pointd.ins

Most of the results for this flow are obtained with η_{max} varying as $\sqrt{s}$. The local variable `btrf` for η_{max} used in subroutine `fdinit` is varied by the following statement in `pointd.ins`.

```
btrf = strfp(2)*sqrt( par(1)*par(2) )
```

6.1.7 postpr.ins

The present module is set up to calculate the following new variables:

1. pressure disturbance, `rp(n,it,iz)`

2. massflow disturbance, `rmass(n,it,iz)`

3. total temperature, `tot(n,it,iz)`

4. $j-th$ covariant component of wall shear stress for steady $(it = 0, iz)$modes, `wshcov(j,iz)`

5. wall heat transfer for steady $(it = 0, iz)$ modes, in $j-th$ covariant direction, `dqwall(j,iz)`

6. maximum amplitude for massflow fluctuation for the reference mode (itr, izr) and corresponding growth rates of other modes at the same location, `ampmax(it,iz)`

7. growth rate profiles ($n = 0, jx$) of ig-th variable at the location where the mass-flow disturbance of the reference mode (itr, izr) is maximum, `grate(n,it,iz,ig)`. Here $ig = 1,..,4$ representing growth rates for $\tilde{\rho}$, $\tilde{T}$, $\tilde{u}^{\xi^1}$, and mass flow $\tilde{m}$, respectively.

6.1.8 Definitions

A sample input file `Definitions` for stability analysis is given in Figure 6.2. Since

```
Stab. eqs. in  rho, T, phy. contr. u in body-intrinsic coordinates
200                 /no. of collocation points -1
10                  /number of constants
13                  /number of parameters
-1  -1  -1  -1  -1 /idsy(nvar) -1 eta no symmetry, 0 symm, 1 antisymm
 0  4  4  4  4     /idbc(nvar) type of boundary conditions
-1                  /idpts=1 Gauss-Lab., 0 b-state grid, -1 use idtrf
4, 0.000, 22., 1.12, 0. /idtrf, atrf, btrf, str3, str4
rho      , 1 /name, z-symmetry of variable 1
T        , 1 /name, z-symmetry of variable 2
u        , 1 /name, z-symmetry of variable 3
v        , 1 /name, z-symmetry of variable 4
w        ,-1 /name, z-symmetry of variable 5
WallProp  0.0000000000000 / con(1), not used
Air-Prop          1120103. / con(2), model # for air properties
Tinfty    54.30000000000  / con(3), Freestream temperature, (Kelvin).
Pinfty    4.082743600e-03 / con(4), Freestream pressure, (atm).
Lref      0.666750000e+00 / con(5), Reference length (m).
Prandtl   0.7200000000000 / con(6), Edge Prandtl number.
BLmodel   311.00000000000 / con(7), Analy. model; mvar,mcoord,mdlddx
Hflux     0.000000000E-04 / con(8), Heat flux  (not used)
npprt     5.000           / con(9), print freq. in steps for profile data
relaxp    0.000           / con(10), 1 dp/dx removed for oscil. modes, 0 NOT
NG1        6.000000000000 / points used to interpolate basic state in xi^{1}
NG2        6.000000000000 / points used to interpolate basic state in xi^{2}
NG3        6.000000000000 / points used to interpolate basic state in xi^{3}
xi^{1}   , 2, 0 /par(1), affects basic state, real
Reinv    , 0, 0 /par(2), appears linear, real
alphar   , 1, 1 /par(3), appears NON-linear, complex(real)
alphai   , 1, 1 /par(4), appears NON-linear, complex(imag)
beta     , 1, 0 /par(5), appears NON-linear, real
omegar   , 0, 1 /par(6), appears linear, complex(real)
omegai   , 0, 2 /par(7), appears linear, complex(imag)
UsqdRT   , 2, 0 /par(8), appears linear, affects bstate; real
xi^{3}   , 2, 0 /par(9), affects basic state, real
```

Figure 6.2: Example Input File `Definitions` for Stability Analysis of Hypersonic Flow Over a Blunt Cone at $M_\infty = 8$.

`idbc` is set to 4, asymptotic boundary conditions are used at farfield. Here a slight logarithmic stretching (`idtrf`=4 with `strfp(3)` =1.12) is used. The wall is at η_{min} = `strfp(1)` = 0, while the farfield boundary at the reference station is located at nondimensional η_{max} = `strfp(2)` = 22. Note that a larger value should be used for `strfp(2)` if homogeneous conditions are applied at farfield. In the next

nvar=5 lines, the names of dependent variables for disturbances and their symmetry property in $\xi^3 = \phi$ directions are defined.

Constants and parameters may be defined by the user to suit a particular application. There are ncon=13 constants and npar=9 parameters. con(1) is not used. con(2) named as Air-prop specifies the thermodynamic model for the air. This is specified as 1120103. (see Appendix A for details). Constants con(3) and con(4) give reference temperature and pressure for nondimensionalization. Their edge values vary with streamwise distance for a blunt cone, unlike those for a sharp cone. Therefore, fixed basic-state freestream values of temperature (tinfty $= T^*_\infty = 54.3\,K$) and pressure (pinfty=p^*_∞ = 0.004082743 $atm.$) are used as reference values. The next constant con(5) is the location of the reference station in arc length (meters) and is named as Lref. In our results, the station at $s^*/R_N = 175$, $s^* = 0.66675m.$ was input as a reference station. Since the 10^0 digit in the Air-Prop model is set to 3, the reference length scale rl, is the boundary-layer length scale $\sqrt{\frac{\nu s^*}{U_\infty}} = 2.8511 \times 10^{-4}\,m$ at the reference station, $s^*/R_N = 175$. Prandtl number is specified as con(6).

The three digits of the real constant blmodel =con(7) specify the following:

10^2 digit — mvar, the choice of dependent disturbance variable. Different choices are described in Section 2.2.6. *The meaning of this flag* mvar *is generic for any application.*

10^1 digit — mcoord, the choice of coordinate system for stability analysis
- 0 — local Cartesian coordinates tangent to the cone surface
- 1 — curvilinear coordinates

10^0 digit — mddlx, the flag for basic-state nonparallel terms
- 0 — parallel basic state, ξ^2 velocity component and (ξ^1, ξ^3) gradients of basic flow are neglected
- 1 — nonparallel basic state, ξ^2 velocity component and (ξ^1, ξ^3) gradients of basic flow are included

In the example, physical contravariant velocities (mvar=3) in curvilinear coordinates (mcoord=1) for a nonparallel basic state (mddlx=1) are chosen. The rest of the constants are self-explanatory by way of comments in the example.

The nine dimensionless parameters defined in this example represent the following:

- par(1) — ξ^1
- par(2) — Reinv, the inverse of the Reynolds number
- par(3) — alphar, real part of the complex wavenumber in ξ^1 direction α_r
- par(4) — alphar, imaginary part of the complex wavenumber in ξ^1 direction α_i
- par(5) — beta, real wavenumber in ξ^3 direction
- par(6) — omegar, real part of complex temporal frequency, ω_r

par(7) omegai, imaginary part of complex temporal frequency, ω_i
par(8) UsqdRT, $U_\infty^2/RT_\infty = \gamma M_\infty^2$
par(9) ξ^3 coordinate, i.e., azimuthal angle ϕ

6.1.9 Parameters

Parameters and tasks of Linear stability analysis to be performed in running LISA are assigned by the user in the input file Parameters. The first example in Figure 6.3 is for calculating local eigenvalues and eigenvectors at a specified ξ^1 location.

```
1                 /task :
3   4             /eigenvalue (set to alpha)
2338.535          /parameter  1 xi^{1} (=re,usually)
4.2761798E-04     /parameter  2 1/re
2.1000000000E-01  /parameter  3 alphar
-3.656700000E-05  /parameter  4 alphai
0.0000000000E+01  /parameter  5 beta
1.9340000000E-01  /parameter  6 omegar
0.0000000000E+01  /parameter  7 omegai
8.9600000000E+01  /parameter  8 U**2/RT
0.0000000000E+01  /parameter  9 xi^{3}
3                 /task
1                 /kdim
6   20    .004    /parameter, steps, step size
-999/END
```

Figure 6.3: Example Input File Parameters for Finding Spatial Eigenvalues and Generating a Table for Varying ω_r at Station $s = s^*/\mathrm{rl} = 2338.535$ for 2D Second-Mode Disturbances.

Here itask=1 is specified as a first task. Parameters 3 and 4, i.e., real and imaginary of wavenumber α are defined as unknown eigenvalues. The streamwise location is specified in par(1) in dimensionless units (in terms of boundary-layer length scale $\mathrm{rl} = \sqrt{\nu s^*/U_\infty}$ at the reference station $s^*/R_N = 175$). The inverse of the Reynolds number at the reference station is given in par(2). The eigenvalues par(3) and par(4) are given initial guesses. The wavenumber in azimuthal direction par(5) is set to zero to study 2D disturbances. The dimensionless frequency $\omega_r = f^* \mathrm{rl}/(2\pi U_\infty)$ is specified as par(6). The imaginary part ω_i is set to zero for calculating spatial growth. The value of par(8) $= \gamma M_\infty^2$ is 89.6 with $M_\infty = 8$. The basic state is axisymmetric and hence par(9)=ϕ is set to zero. After this task is finished, the next task itask=3 generates a table of solutions for different ω_r. The step size for table may be assigned a negative value.

The second example (Figure 6.4) first performs (itask=2) at the initial station $s = s^*/\mathrm{rl} = 1400$. Hence eigenvalues and the corresponding eigenfunctions are calculated

```
2                   /task :
3   4               /eigenvalue (set to alpha)
1400.0              /parameter  1 xi^{1} (=re,usually)
4.2761796E-04       /parameter  2 1/re
2.1000000000E-01    /parameter  3 alphar
-3.656700000E-05    /parameter  4 alphai
0.0000000000E+01    /parameter  5 beta
1.9340000000E-01    /parameter  6 omegar
0.0000000000E+01    /parameter  7 omegai
8.960000000E+01     /parameter  8 U**2/RT
0.000000000E+01     /parameter  9 xi^{3}
3                   /task
1                   /kdim
1    78    30       /parameter, steps, step size
-999/END
```

Figure 6.4: Example Input File **Parameters** for Finding Spatial Eigenvalues and Generating a Table for Varying $\xi^1 = s$ at the Fixed Frequency of $\omega_r = 0.1934$ for 2D Second-Mode Disturbances.

at this station. Eigenfunctions are written out to the output file `0.function`. Then a series of solutions are generated (`itask=3`) at the fixed values of other *specified* parameters for several stations by varying s. The file `0.function` is overwritten each station so that only the eigenfunctions at the last station will be saved.

6.1.10 Control

An example input file `Control` for a nonlinear PSE run for two-dimensional disturbances is given in Figure 6.5.

Comments provide brief explanations of entries in the file. There are five distinct modes to be included in this calculation. The initial mode is the fundamental mode `(1,0)` with dimensionless frequency `tpar`. Mode control `mgr` for this mode is 2, the default value. This mode is calculated by the local linear stability analysis with `itask=2` and read in by PSH. Its initial amplitude for maximum $\tilde{u}^{\xi^1}$ disturbance (i.e., `normv` is set to 3) is specified as 2×10^{-4}. Three other modes are harmonics of this initial mode, to be created by the PSH through subroutine `newmodes` if the nonlinear forcing terms are greater than the specified `thron` of 5×10^{-9}. The phase-locking condition will be used for these harmonics, as `mgr`'s are set to 1. The fifth `(0,0)` mode is meanflow distortion. `mgr` for this mode is also set to 1.

```
Transition in Hypersonic Flow over a Blunt Cone at Mach 8
1   26   1  20  /Initial and final stations; station change; step size
0               /inlt, nterms treated implicitly for 1, explicitly for 0
250             /Iterations (linear <= 1), max step divisions
.1934    0      /tpar (frequency), zpar (beta)
1  0            /itr, izr (reference mode).
5e-9  5e-10     /Thresholds on(rhs)/off(amp)
14              /Number of mode control lines
0 0    2        /T-index, Z-index, include this mode.
0 1   -1        /T-index, Z-index, disregard this mode.
0 2   -1        /T-index, Z-index, disregard this mode.
2 0    1        /T-index, Z-index, include with phase lock.
1 1   -1        /T-index, Z-index, disregard this mode.
1 2   -1        /T-index, Z-index, disregard this mode.
2 1   -1        /T-index, Z-index, disregard this mode.
2 2   -1        /T-index, Z-index, disregard this mode.
3 0    1        /T-index, Z-index, include with phase lock.
3 1   -1        /T-index, Z-index, disregard this mode.
3 2   -1        /T-index, Z-index, disregard this mode.
4 0    1        /T-index, Z-index, include with phase lock.
4 1   -1        /T-index, Z-index, disregard this mode.
4 2   -1        /T-index, Z-index, disregard this mode.
1               /Number of initial modes
1 0 2  2.e-4    0./T-index, Z-index, mgr, amp, phase
```

Figure 6.5: Example Input File `Control` for the Nonlinear PSE Run for Two-Dimensional Second-Mode Disturbance.

6.2 Hypersonic Flow over a Sharp Cone at Angle of Attack

6.2.1 metric.ins

The disturbance coordinate system is *nonorthogonal* and almost identical to the algebraic one that the AFWAL PNS code uses. It is given by:

$$\begin{aligned} x^1 &= \xi^1 \\ x^2 &= R\sin(\xi^3 - \pi) &&= -R\sin(\xi^3) \\ x^3 &= R\cos(\xi^3 - \pi) &&= -R\cos(\xi^3) \\ R &= r_b(\xi^1, \xi^3) + \xi^2 \end{aligned} \tag{6.1}$$

where ξ^1 is the distance measured along the body axis, ξ^3 is the circumferential angle measured in radians from the windward meridian in the $-\xi^1$ direction, r_b is the local body radius, and ξ^2 is the radial distance from the body surface. See Figure 6.6. The similarity between the AFWAL PNS algebraic grid and the coordinate system used for the stability analysis is intentional and enables the basic state to be interpolated with greater ease and accuracy.

The user can either use this coordinate system or a local Cartesian coordinate system oriented tangent to the body surface by setting the 10^1, or 10's digit, in `blmodel` to `1` or `0`, respectively, in the Definitions file. (This digit is named `icoord` in the `metric.ins` file.) In the latter case, any curvature of the body surface will be neglected.

6.2.2 gridbs.ins

Subroutine `gridbs` assigns the basic state grid simply by reading it in along with the basic state solution. Auxiliary routines that `gridbs` calls to accomplish this are described more fully in Appendix C. Of these, three are specific to the basic state flow solver used. They are: `bsiois`, `bpstda`, and `scale` .

Subroutines `bsiois` and `bpstda` read in the station data and flow profiles, respectively, from FORTRAN unformatted PLOT3D files. These files contain consecutive axial planes of data created by the AFWAL PNS code.

The AFWAL Parabolized Navier-Stokes (PNS) code has been developed by several groups since 1979 and has been documented by Kaul and Chaussee (1983); Stalnaker, Nicholson, Hanline, and McGraw (1986); and Rajendran (1989). It uses a space-marching technique and body-fitted coordinates to integrate the PNS equations downstream of data specified in an initial axial plane $x^1 = x_0^1$.

The permissible AFWAL PNS body-fitted coordinates are of the form:

$$x^1 = x^1(I) \tag{6.2}$$

$$x^2 = x^2(J, K) \tag{6.3}$$

$$x^3 = x^3(J, K) \tag{6.4}$$

where x^1 is the body axis (see Figure 6.6). This means that the solution is marched

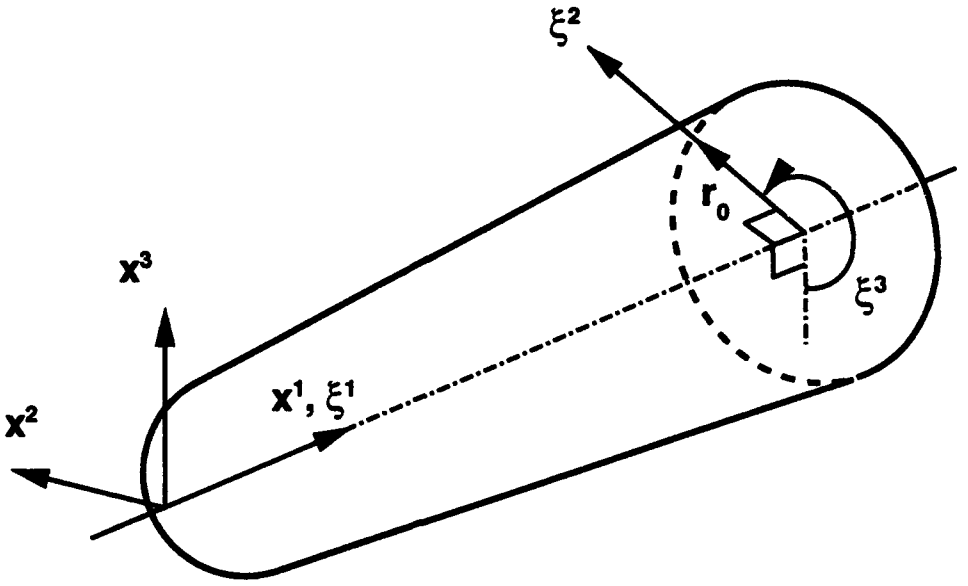

Figure 6.6: Coordinate System Used in the Stability Analyses.

in planes perpendicular to the body axis. Hence, the body-fitted coordinates are nonorthogonal whenever the slope of the body in the axial direction is nonzero. This is certainly the case for any cone of nonzero half-angle.

Virtually the same coordinate system (Section 6.2.1) is used for the stability analyses to simplify the problem of interpolating the basic state solution onto the disturbance grid. The interpolation scheme does, however, place restrictions on the transformation between the basic state coordinates (K, I, J) and the disturbance coordinates (ξ^1, ξ^2, ξ^3). These restrictions are discussed in Appendix C .

The PLOT3D files must be created according to the formats used by the following routines to read them.

bsiois

Subroutine `bsiois` transforms the Cartesian grid points in the body surface to cylindrical coordinates and stores the required data in the matrix `stndat` .

The relevant `open`, `read` and station data assignment statements are:

```
      one = 1.
      pi  = acos(-one)

C  Open the PLOT3D grid file ...
```

```
      open(unit=ubsf(1),file=bsfn(1),access='sequential',status='old',
     |     form='unformatted')
      read(ubsf(1)) idim,jdim,kdim

C  Open the PLOT3D solution file ...

      open(unit=ubsf(2),file=bsfn(2),access='sequential',status='old',
     |     form='unformatted')
      read(ubsf(2)) id1,id2,id3

      if (id1.ne.idim .or. id2.ne.jdim .or. id3.ne.kdim) then
         write(6,98) idim,jdim,kdim,id1,id2,id3
   98    format(//,' ',72('*'),//,3X,'The dimensions of the basic ',
     |          'state grid and solution files are not the same.',//,
     |          3X,'GRID:',5X,I3,' x ',I3,' x ',I3,/,3X,'SOLUTION:',
     |          1X,I3,' x ',I3,' x ',I3,//,' ',72('*'))
         stop
      end if
C
C  Free-stream Re assumed to be based on free-stream speed of sound and
C  NOT the free-stream flow speed.
c
      read(ubsf(2)) fsmach,attack,fsre,fstemp

      istn=0
      nk=0
      do 300 k=1,kdim
         read(ubsf(1),iostat=ie(1)) ((xb     ,i=1,idim),j=1,jdim),
     |                              ((yb(i,j),i=1,idim),j=1,jdim),
     |                              ((zb(i,j),i=1,idim),j=1,jdim)
C
C  Don't repeatedly store data for the same station ...
C
         if (k.gt.1.and.xb.eq.xbo) go to 400

         if (ie(1).eq.0.and.ie(2).eq.0) then

            xbo  = xb
            nk   = nk+1

            if (k.eq.1) then
C
C  Target value for XI3BSC computed from YB(1,1) and ZB(1,1).  The
C  branch used for the arctangent will be chosen so that XI3(K=1,J=1)
C  will be closest to this target value.
```

```
C
               write(6,903) xb,yb(1,1),zb(1,1)
  903          format(/,' ','Converting PLOT3D grid file from Cartesian'
     |              ,' coordinates to cylindrical coordinates.',/,' ',
     |              'Must choose branch of arctangent.  Enter the ',
     |              'circumferential angle for point NEAR',/,' ','(not ',
     |              ' necessarily coincident with) the point:',/,' ',
     |              'XB = ',E12.6,' YB(1,1) = ',E12.6,' ZB(1,1) = ',
     |             E12.6,' => ',$)
               read(5,*) xi3lst(0)
               xi3lst(0) = pi
               w = one
            else
C
C  For k>1, choose the branch of the arctangent so that the computed
C  value of xi3(k,1) is near xi3(k-1,1).
C
               xi3lst(0) = xi3lst(1)
               w = 0.5

            do 100 j=1,jdim
               istn   = istn+1
               xi1bsc = xb
               xi3bsc = atan2(-yb(1,j),-zb(1,j))
C
C  Choose the branch of the arctangent so that xi3bsc is near the
C  average value of xi3(k-1,j) (stored in xi3lst(j)) and xi3(k,j-1)
C  (stored in xi3lst(j-1)).
C
               branch = nint((w*(xi3lst(j)+xi3lst(j-1))-xi3bsc)
     |                        /(2.*pi))
               xi3bsc = xi3bsc + 2*branch*pi
               if (k.eq.1) write(6,*) 'xi3bsc=',xi3bsc,' branch=',branch

               stndat( 0,istn) = 1+idim*((k-1)*jdim+(j-1))
               stndat( 1,istn) = xi1bsc
               stndat( 2,istn) = idim
               stndat( 3,istn) = xi3bsc
               stndat( 4,istn) = k

C  Local body radius ...

               stndat( 5,istn) = sqrt(yb(1,j)**2+zb(1,j)**2)

C  Local shock stand-off distance measured from body axis ...
```

```
              stndat( 6,istn) = sqrt(yb(idim,j)**2+zb(idim,j)**2)

              xi3lst(j) = xi3bsc
  100       continue

        else

C  Write diagnostics for read errors ..

        end if

  300 continue

  400 close(ubsf(1))
      close(ubsf(2))
```

where

`xi1bsc`	$\equiv$	ξ^1
`xi3bsc`	$\equiv$	ξ^3
`yb`	$\equiv$	x^2
`zb`	$\equiv$	x^3
`idim`	=	number of basic state grid points in ξ^2
`jdim`	=	number of basic state grid points in ξ^3
`kdim`	=	number of basic state grid points in ξ^1

Typically, the AFWAL PNS code will store the flow field in the range $\pi \leq \xi^3 \leq 2\pi$ when the flow and body are symmetric about the $x^3 = z$ axis.[1] For this half-interval, the user should enter a value of ξ^3 near $3 \approx \pi$ to select the right branch of the arctangent needed to convert the PLOT3D Cartesian coordinates to cylindrical coordinates.

Subroutine `bsiois` also sets the scales for the basic state solution. The PLOT3D `bsiois` sets the reference length to 1" for ξ^1 and ξ^2. It then converts this reference length to units of meters before storing it. The scale for ξ^3, the azimuthal angle measured around the body axis, is simply set to 1. Also, `bsiois` assumes that the reference temperature for the basic state calculation is assigned to the variable normally reserved for time on the second line in the PLOT3D file:

[1]We did not realize this at the beginning of our analyses with the PNS code. In retrospect, it would have been better to define $\xi^3 = 0$ along the windward meridian so that the computed half-interval would be $0 \leq \xi^3 \leq \pi$. This could be changed by modifying the file `metric.ins` .

```
C  Open the PLOT3D solution file ...

      open(unit=ubsf(2),file=bsfn(2),access='sequential',status='old',
     |     form='unformatted')
      read(ubsf(2)) idim,jdim,kdim
C
C  Free-stream Re assumed to be based on free-stream speed of sound and
C  NOT the free-stream flow speed.
c
      read(ubsf(2)) fsmach,attack,fsre,fstemp
```

Although this temperature is assumed to be measured in degrees Rankine (re: AFWAL PNS), subroutine `bsiois` stores it in Kelvin. Finally, the Reynolds number which is read in is based on the free-stream speed of sound, but subroutine `bsiois` stores the Reynolds number based on the free-stream velocity. All of these conventions can easily be changed by modifying the corresponding assignment statements in subroutine `bsiois` .

bpstda

Subroutine `bpstda` transforms the Cartesian grid to cylindrical coordinates and the Cartesian velocity components to the contravariant components in the stability coordinates. It also computes the temperature from the total energy per unit volume that is read in and, finally, writes out all the transformed data to the direct access file station by station.

The relevant `open` and `read/write` statements for the PLOT3D files are:[1]

```
      parameter (ibs1=501,ibs2=19,nbsvar=5)

      gamma = realv(13)
      gmm1  = gamma-one
      ggmm1 = gamma*gmm1

      open(unit=ubsf(1),file=bsfn(1),access='sequential',status='old',
     |     form='unformatted')
      open(unit=ubsf(2),file=bsfn(2),access='sequential',status='old',
     |     form='unformatted')

      read(ubsf(1),iostat=ie(1)) idim,jdim,kdim
      read(ubsf(2),iostat=ie(2)) id1,id2,id3

      if (id1.gt.ibs1 .or. id2.gt.ibs2) then
         write(6,99) id1,id2,ibs1,ibs2
 99      format(//,' ',72('*'),//,3X,'The dimensions of the basic ',
```

[1]The real variables in the PLOT3D files must be *single precision*.

```
     |          'state grid and solution files are too big.',//,
     |          3X,'FILES: ',5X,I3,' x ',I3,/,3X,'MEMORY:',
     |          1X,I3,' x ',I3,//,' ',72('*'))
        stop
      end if

      read(ubsf(2),iostat=ie(2)) fsmach,attack,fsre,fstemp

      istn=0
      do 300 k=1,kdim
      read(ubsf(1),iostat=ie(1)) ((xb     ,i=1,idim),j=1,jdim),
     |                           ((yb(i,j),i=1,idim),j=1,jdim),
     |                           ((zb(i,j),i=1,idim),j=1,jdim)
      read(ubsf(2),iostat=ie(2)) (((qb(i,j,nv),i=1,idim),j=1,jdim),
     |                                          nv=1,nbsvar)

      if (ie(1).eq.0.and.ie(2).eq.0) then

         do 120 j=1,jdim
            istn = istn+1
            key  = nint(stndat(0,istn))
C
C  Rotate into cylindrical coordinates (contravariant components).
C  Assume that the radial lines are STRAIGHT!
C  This step is essentially the stability analysis grid generation.
C  Also, scale the velocity components with the free-stream flow speed
C  instead of the free-stream speed of sound.
C
            kb=max(min(k,kdim),2)
            d1rd1= (stndat(5,(kb-1)*jdim+j)-stndat(5,(kb-2)*jdim+j))
     |           /(stndat(1,(kb-1)*jdim+j)-stndat(1,(kb-2)*jdim+j))
            d1rd3 = 0

            do 110 i=1,idim
C              eta     = atan2(-yb(i,j),-zb(i,j))
               rb      = sqrt(yb(i,j)*yb(i,j)+zb(i,j)*zb(i,j))
               bvel(1) = qb(i,j,2)/qb(i,j,1)/fsmach
               bvel(2) = qb(i,j,3)/qb(i,j,1)/fsmach
               bvel(3) = qb(i,j,4)/qb(i,j,1)/fsmach
               qb(i,j,2) = (ggmm1/qb(i,j,1))*(qb(i,j,5)-0.5d0/qb(i,j,1)
     |                     *(qb(i,j,2)**2+qb(i,j,3)**2+qb(i,j,4)**2))

C  Contravariant velocity components in cylindrical coordinates ...

               qb(i,j,3) = bvel(1)
```

```
                qb(i,j,4) = (yb(i,j)*bvel(2)+zb(i,j)*bvel(3))/rb
                qb(i,j,5) = (zb(i,j)*bvel(2)-yb(i,j)*bvel(3))/rb/rb

C  Contravariant velocity components in cylindrical coordinate system
C  with xi^2 = 0 at surface ...

                qb(i,j,4)= -d1rd1*qb(i,j,3) + qb(i,j,4) - d1rd3*qb(i,j,5)

                write(uda(1),rec=key+i-1) rb,(qb(i,j,nv),nv=1,5)
  110         continue
  120       continue
        end if

  300 continue
```

6.2.3 bstate.ins

Subroutine `bstate` reads in and interpolates the basic state profiles from the direct access file created by subroutine `gridbs`. The method by which it does this is independent of the input and output routines required to read in the basic state, and so need not be changed for different basic state flow solvers. It is described more fully in Appendix C.

6.2.4 btrans.ins

This file specifies the relationship between the basic state variables that are used in the stability equations and the basic state variables that are used in subroutine `bstate`. As described in Section 2.2.4, this relationship may be selected by specifying density, temperature, and one of three sets of velocity components. These three sets of velocity components are:

1. Cartesian components in an inertial system.
2. Physical contravariant components in the disturbance coordinates.
3. Contravariant components in the disturbance coordinates.

The user specifies the set of velocity components for `btrans` by setting the integer variable `ibvel = 0,1,` or 2, respectively.

Although the AFWAL PNS solution and the PLOT3D files that it creates use Cartesian velocity components, the velocity vector in `btrans` is specified in terms of the contravariant components (`ibvel = 2`), because the AFWAL PNS code runs axisymmetric calculations fully three-dimensional with a coarse[1] point distribution

[1] Assuming that the reference coordinates used for the PNS solution are cylindrical - an option that the AFWAL PNS user can select.

in the circumferential direction. The Cartesian velocity components, since they vary *sinusoidally* around the circumference of an axisymmetric body at zero angle of attack, would not be interpolated very accurately on a coarse grid. The contravariant components, which are *invariant* in the circumferential direction, *can* be interpolated accurately.

Two different options are also implemented to control some of the terms in the stability equations. The ξ^2 contravariant velocity component and all ξ^1 and ξ^3 derivatives of the basic state can be set identically to zero by setting the 10^0, or 1's place, in `blmodel` to zero in the Definitions file. (`blmodel = 0.`) Alternatively, all of these terms are retained in the analysis for a nonzero value of this same digit in `blmodel` .

Subroutine `btrans` also sets the the thermophysical properties for the disturbance analyses. This is done through calls to subroutine `therm`, described in Appendix A . The user can select which constitutive equations `therm` must use for the properties by setting the second constant, `Air-Prop`, in the `Definitions` file to a value as described in Appendix A .

6.2.5 boundary.ins

The boundary conditions which are implemented in `boundary.ins` are homogeneous. No options enable the use of asymptotic boundary conditions in the far-field, but this could be changed. Also, the disturbance is also assumed to satisfy the no-slip and kinematic boundary conditions at the body surface and the disturbance in the derivative of the temperature with respect to ξ^2 is set identically to zero.

6.2.6 pointd.ins

The radial point distribution can be made to vary with ξ^1 by modifying the grid parameters in `pointd.ins` . The current setting makes `chy(jx)`, the point furthest from the surface, vary according to:

$$\texttt{btrf} = \texttt{strfp(2)} * \texttt{sqrt(par(1)} * \texttt{par(2))} \tag{6.5}$$

i.e., like the square root of the ratio of ξ^1/ξ_0^1, where ξ_0^1 is the value of ξ^1 at the station where the disturbance reference length $= \xi_0^{1*}/\sqrt{Re_{\xi_0^1}} = \xi_0^{1*}/xi_0^1 = \xi^{1*} * \texttt{par(2)}$.

6.2.7 postpr.ins

No `postpr.ins` file was written for this application. However, the corresponding file for the blunt cone application (Section 6.1.7) could be modified and used here as well. Data for the figures shown in the final report were taken directly from the `0.table` file and `0.summary` file created by the local stability code and PSE code, respectively.

6.2.8 Definitions

The `Definitions` file is virtually identical to the one described for the blunt cone application (Section 6.1.8). The differences are in the numerical values of the reference data that are input and the controls for the boundary conditions. There are no controls for the sharp cone boundary conditions (see Section 6.2.5). The print control constant `npprt` is also not present because a `postpr.ins` file was not written for the sharp cone application. Finally, the meanings of the parameters are also different because the coordinate systems are different. (See Sections 6.1.1 and 6.2.1, respectively.) While the *specific* meaning of the parameters and some constants is different for different coordinate systems, their general definitions are the same if interpreted in terms of aribtrary (ξ^1, ξ^2, ξ^3).

6.2.9 Parameters and Control

The `Parameters` and `Control` files can be specified for the sharp cone application just as they were for the blunt cone application. See Sections 6.1.9 and 6.1.10, respectively.

Chapter 7

References

Bertolotti, F. P. 1991 "Linear and Nonlinear Stability of Boundary Layers with Streamwise Varying Properties," *Ph.D. Thesis*, The Ohio State University, Columbus, Ohio.

Borisenko, A. I., and Tarapov, I. E. 1979 "Vector and Tensor Analysis with Applications." Translated from the Russian by R. A. Silverman. Published by Dover Publications, Inc., New York.

Esfahanian, V. 1991 "Computation and stability analysis of laminar flow over a blunt cone in hypersonic flows", Ph. D. Thesis, The Ohio State University, Columbus, Ohio.

Gregory, N., Stuart, J.T., and Walker, W.S., 1955, "On the Stability of Three-Dimensional Boundary Layers with Application to the Flow Due to a Rotating Disk," *Phil. Trans. R. Soc. London,* Vol. 248, pp. 155-199.

Herbert, T., Stuckert, G. k., and Lin, N. 1993 "Method for Transition Prediction in High-speed Boundary Layers," Final Report for Contract No. F33615-90-C-3009, Wright Patterson Air Force Base, Dayton, Ohio.

Herbert, T. 1991 "Boundary-Layer Transition - Analysis and Prediction Revisited," *AIAA-91-0737.*

Herbert, T. 1990 "Linear.x - A Code for Linear Stability Analysis," in *Instability and Transition*, Eds. M.Y. Hussaini and R.G. Voigt, pp. 121-144, Springer Verlag.

Kachanov, Y.S., and Trararykin, O.J. 1990 "The Experimental Investigation of Stability and Receptivity of a Swept Wing Flow," in: *Laminar-Turbulent Transition*, IUTAM Symp., Toulouse, France. Springer-Verlag

Kaups, K. and Cebeci, T., 1977, "Compressible Laminar Boundary Layers on Swept

and Tapered Wings," *Journal of Aircraft*, Vol. 14, pp. 661-667.

Lancaster, P., and Salkauskas, K., 1988, "Curve and Surface Fitting - An Introduction," Academic Press Ltd., San Diego, CA 92101

Mack, L. M. 1984 "Boundary-layer Linear Stability Theory," in: *Special Course on Stability and Transition of Laminar Flows, AGARD Report No. 709*

Rizk, Y. M., Scott, J. N., and Newman, R. K., 1983 "Numerical Solution of Viscous Supersonic Flows in the Vicinity of Embedded Subsonic or Axially Separated Regions, User's Manual for the Three-Dimensional Viscous Nose Tip Code," AFWAL-TR-83-3113, Volume II, Flight Dynamics Laboratory, Wright Research and Development Center, Air Force Systems Command, Wright-Patterson Air Force Base, Ohio 45433-6553.

Stalnaker, J. F., Nicholson, L. A., Hanline, D. S., and McGraw, E. H., 1986 "Improvements to the AFWAL Parabolized Navier-Stokes Code Formulation," AFWAL-TR-86-3076, Flight Dynamics Laboratory, Wright Research and Development Center, Air Force Systems Command, Wright-Patterson Air Force Base, Ohio 45433-6553.

Stetson, K. F., and Kimmel, R.L. 1992 "On Hypersonic Boundary-Layer Stability," *AIAA-92-0737*

Stetson, K. F., Thompson, E. R., Donaldson, J. C., and Siler, L. G., 1984, "Laminar Boundary Layer Stability Experiments on a Cone at Mach 8, Part 1: Sharp Cone," *AIAA-83-1761*

Stetson, K. F., Thompson, E. R., Donaldson, J. C., and Siler, L. G., 1984, "Laminar Boundary Layer Stability Experiments on a Cone at Mach 8, Part 2: Blunt Cone," *AIAA-84-0375*

Stetson, K. F., Thompson, E. R., Donaldson, J. C., and Siler, L. G., 1985, "Laminar Boundary Layer Stability Experiments on a Cone at Mach 8, Part 3: Sharp Cone at Angle of Attack," *AIAA-85-0492*

Stuckert, G.K., Herbert, T., and Esfahanian, V. 1993 "Stability and Transition on Swept Wings," *AIAA-93-0078.*

Walatka, P. P., Buning, P. G., Pierce, L., and Elson, P. A., 1990 "PLOT3D User's Manual," *NASA Technical Memorandum 101067*, NAS Documentation Center, M/S 258-6, NASA Ames Research Center, Moffet Field, CA 94035-1000. Report UTHl-74, University of Technology, Dept. Aero. Eng.

Appendix A

Constitutive Equations

The constitutive relations close the system of flow equations by specifying how the properties of the gas depend on its state. A variety of relations for each required property of *air* are implemented in subroutine `therm`. `therm` computes the *dimensional* properties, scales them, and then returns *dimensionless* data. There is no need to compute and specify a long list of dimensionless constants. Instead, the input of a short list of information is sufficient to determine all the scales.

The user selects the relations and nondimensionalization by assigning a value to a seven digit integer model number named `mdlair`. `mdlair` is passed as the first argument to the subroutine `therm`. Each digit of `mdlair`'s base 10 representation is itself a model number for a particular constitutive relation. Hence, there may be up to 10 different models for each property.

The lowest order digit, 10^0, or 1's place, in `mdlair` specifies how to compute the scales used to nondimensionalize the properties. Currently, any of four different choices may be selected. They are all biased towards compressible flow applications because they all require that the reference temperature and either the Mach number or $\frac{U_\infty^{*2}}{R^* T_\infty^*}$ also be input. With this additional data, and assuming that in its reference state air behaves as an ideal gas, `therm` can compute the reference speed of sound, reference velocity U_∞^*, and reference viscosity μ_∞^*. How `therm` proceeds to compute the remaining scales depends on which of the four options the user selects from Table A.1.

The first two options in Table A.1 are available for those situations when most, but not all, of the dimensionless parameters are known. These are important cases because all the properties that `therm` computes, except for the density, vary only with temperature. Subroutine `therm` can thus compute the (dimensional) reference values of all the properties except for the density *knowing only the reference temperature.* Using these reference values as scales, `therm` can subsequently nondimensionalize *all* the constants appearing in all the constitutive equations except for the equation of state. The equation of state for an ideal gas, on the other hand, can easily be coded in dimensionless form and depends on only one dimensionless constant: $U_\infty^{*2}/(R^* T_\infty^*)$.

10^0 digit	Selection
0	Input the reference pressure (atm) and the inverse of the Reynolds number. `therm` computes the reference density (kg/m^3) from the ideal gas law for air. It then determines the reference length (m) from the inverse of the Reynolds number.
1	Input the reference length (m) and the inverse of the Reynolds number. `therm` computes the reference density (kg/m^3) from the inverse of the Reynolds number. It then determines the pressure (atm) from the ideal gas law for air.
2	Input the reference pressure (atm) and the reference length (m). `therm` computes the reference density (kg/m^3) from the ideal gas law for air. It then computes the inverse of the Reynolds number.
3	Input the reference pressure (atm) and a length c (m). `therm` computes the reference density (kg/m^3) from the ideal gas law for air. It then computes the inverse of the Reynolds number Re_c based on c. Finally, *the reference length is set to* $c/\sqrt{Re_c}$ *and a new Reynolds number* $= \sqrt{Re_c}$ *based on this reference length is computed.*

Table A.1: Scales selection

10^1 digit	Selection
0	$\frac{U_\infty^{*2}}{R^* T_\infty^*}$ is input
1	Mach number is input.

Table A.2: Parameter selection

Hence, for fixed values of this constant[1] and for fixed reference temperature, all the dimensionless constitutive equations are *independent* of the reference pressure or reference length that the user inputs and consequently, *so is the dimensionless solution for the flow.*

The options that set the 10^0 digit = 0 or 1 are typically used when studying the stability of the flat plate compressible boundary layer with similar velocity profiles. The options that set the 10^0 digit = 2 or 3 are used when the user does not know the Reynolds number and wants `therm` to compute the value by using the chosen constitutive equations.

Continuing with the other digits in `mdlair`, the 10^1 digit, or 10's place, indicates whether $\frac{U_\infty^{*2}}{R^* T_\infty^*}$ or the Mach number is input. See Table A.2.

The 10^2 digit, or 100's place, indicates the equation of state. The equation of state specifies the density as a function of pressure and temperature. The density is nondimensionalized by the reference density, the temperature is nondimensionalized by the reference temperature, and the pressure is nondimensionalized by the reference density multiplied by the square of the reference velocity. Table A.3 gives the different models that the user can select from.

The 10^3 digit, or 1000's place, indicates the constitutive equation for the specific heat at constant pressure. The specific heat is nondimensionalized by the gas constant. The three models which have been implemented are given in Table A.4.

The 10^4 digit, or 10,000's place, indicates the model for the first coefficient of viscosity. Table A.6 identifies the four different relations which have been implemented. The dynamic viscosity is nondimensionalized with its value at the reference temperature.

The 10^5 digit, or 100,000's place, indicates the model for the *second* coefficient of viscosity. Currently, only two models have been implemented. See Table A.7. It is nondimensionalized by the value of the first viscosity coefficient evaluated at the reference temperature.

[1] which is either input or directly calculable knowing the input Mach number and reference temperature

10^2 digit	Selection
0	$\rho^* \equiv \rho^*_\infty$: incompressible flow.
1	$\rho^* = \frac{p^*}{R^* T^*}$: ideal gas. The gas constant R^* used for air is 287 $J/(kg \cdot K)$.

Table A.3: Equation of state selection

10^3 digit	Selection
0	$c_p^* = \frac{\gamma R^*}{\gamma - 1}$, where $\gamma \equiv 1.4$ is a constant. (Calorically perfect gas.)
1	$c_p^* = a_0^* + a_1^* T_\infty^* + a_2^* T_\infty^{*2} + a_3^* T_\infty^{*3} + a_4^* T_\infty^{*4}$, the dimensional constants having been determined by Fabio Bertolotti using a least squares fit to a set of experimental data in the temperature range $100K \leq T^* \leq 1600K$. The dimensional constants for T^* in K and c_p in $J/(kg \cdot K)$ are listed in Table A.5.

Table A.4: Specific heat constitutive relations

i	a_i^* $(c_p^*, J/(kg \cdot K))$	b_i^* $(\mu^*, N \cdot s/m^2)$	c_i^* $(\kappa^*, W/(m \cdot K))$
0	1.0581839 x 10^{+3}	-1.5616320 x 10^{-7}	-1.3058847 x 10^{-3}
1	-4.5245471 x 10^{-1}	7.9579899 x 10^{-8}	1.0991345 x 10^{-4}
2	1.1413454 x 10^{-3}	-6.9301497 x 10^{-11}	-6.8469791 x 10^{-8}
3	-7.9573904 x 10^{-7}	4.0681578 x 10^{-14}	3.3270833 x 10^{-11}
4	1.9108582 x 10^{-10}	-9.1824860 x 10^{-18}	-5.3978665 x 10^{-15}

Table A.5: Least squares polynomial coefficients for thermophysical data

10^4 digit	Selection
0	$\mu^* = \mu^*_\infty$: constant dynamic viscosity.
1	$\mu^* = \mu^*_\infty T^*/T^*_\infty$: linear variation.
2	$\mu^* = \mu^*_r \left(\frac{T^*}{T^*_r}\right)^{3/2} \left(\frac{1+c^*_r/T^*_r}{T^*/T^*_r+c^*_r/T^*_r}\right)$: Sutherland's law, where $c^*_r = 110.4\ K$ and $\mu^*_r = 1.719\text{x}10^{-5} N \cdot s/m^2$ at $T^*_r = 273.1K$.
3	$\mu^* = b^*_0 + b^*_1 T^*_\infty + b^*_2 T^{*2}_\infty + b^*_3 T^{*3}_\infty + b^*_4 T^{*4}_\infty$, the dimensional constants having been determined by Fabio Bertolotti using a least squares fit to a set of experimental data in the temperature range $100K \leq T^* \leq 1600K$. The dimensional constants for T^* in K and μ in $N \cdot s/m^2$ are listed in Table A.5.

Table A.6: Constitutive relations for the first coefficient of viscosity

10^5 digit	Selection
0	$\lambda^* = -\mu^*$
1	$\lambda^* = -\frac{2}{3}\mu^*$

Table A.7: Constitutive relations for the second coefficient of viscosity

10^6 digit	Selection
0	$\kappa^* = \frac{Pr}{\mu^*_\infty c^*_{p\infty}} =:$ constant thermal conductivity.
1	$\kappa^* = \frac{Pr}{\mu^* c^*_p} =:$ constant Prandtl number.
2	$\kappa^* = \kappa^*_r \left(\frac{T^*}{T^*_r}\right)^{3/2} \left(\frac{1+s^*_r/T^*_r}{T^*/T^*_r+s^*_r/T^*_r}\right)$: Sutherland's law, where $s^*_r = 194.4\ K$ and $\kappa^*_r = 2.373\text{x}10^{-2} W/(m \cdot K)$ at $T^*_r = 273.1K$.
3	$\kappa^* = c^*_0 + c^*_1 T^*_\infty + c^*_2 T^{*2}_\infty + c^*_3 T^{*3}_\infty + c^*_4 T^{*4}_\infty$, the dimensional constants having been determined by Fabio Bertolotti using a least squares fit to a set of experimental data in the temperature range $100K \leq T^* \leq 1600K$. The dimensional constants for T^* in K and κ in $W/(m \cdot K)$ are listed in Table A.5.

Table A.8: Constitutive relations for the thermal conductivity

Finally, the 10^6 digit, or $1,000,000$'s place, indicates the model for the thermal conductivity. The three choices are given in Table A.8. The thermal conductivity is nondimensionalized by the product of the gas constant and the value of the first coefficient of viscosity evaluated at the reference temperature.

Appendix B

Asymptotic Boundary Conditions

The use of asymptotic boundary conditions in solving the stability equations (PSH and LSH) allows the farfield boundary to be located closer to the wall. Hence fewer grid points are required for adequate resolution. It also eliminates oscillations in the phase of the shape function at the farfield boundary observed in solutions obtained with homogeneous boundary conditions at a truncated normal distance. The boundary conditions are based on the local decay rate in normal direction of the inviscid solutions (Mack 1984). The mean flow velocity is assumed to be uniform locally, which is a reasonable assumption provided the boundary is located away from the shock and about 1.5 time the boundary-layer thickness from the wall.
In body-intrinsic coordinates

$$\begin{aligned} \xi^1 &= s; \text{arc-length along the body} \\ \xi^2 &= \eta; \text{normal distance from the body surface} \\ \xi^3 &= \phi; \text{azimuthal angle} \end{aligned}$$

Wave numbers are defined as

$$\begin{aligned} \alpha &= \frac{\partial \theta}{\partial s} \\ \beta &= \frac{\partial \theta}{\partial z} \\ n_\phi &= \frac{\partial \theta}{\partial \phi} \end{aligned}$$

and the frequency is defined as

$$\omega = \frac{\partial \theta}{\partial t}.$$

The normal-mode solution for pressure disturbance in inviscid uniform freestream is given in Mack (1984). It behaves like

$$\tilde{p} \sim e^{-\sqrt{(\alpha^2+\beta^2)\left(1-\overline{M_e}^2\right)}\, y}$$

where $\overline{M_e}$ is the relative (complex) Mach number in the freestream given by

$$\overline{M_e} = \frac{(\alpha U_e + \beta W_e - \omega)\, M_e}{\sqrt{(\alpha^2+\beta^2)\, T_e}}$$

Here U and W be mean-flow velocity in streamwise x and z directions, respectively. The subscript e denotes freestream values. Other disturbance variables, $\tilde{T}$, $\tilde{u}$, $\tilde{v}$ and $\tilde{w}$ should decay (in y) with the same exponential rate.

The above equations are are derived by Mack in *Cartesian* coordinates. A Cartesian coordinate may be constructed locally at the cone surface so that $(x, y, z = 0)$ directions align with (s, η, ϕ) of the axisymmetric body intrinsic coordinates. Here, $\delta z = r\delta\phi$ is the local rectilinear direction and r is the radius from the cone axis to the boundary point. The wave number n_ϕ (integer number of waves) to be used with $\xi^3 = \phi$ can be related to the rectilinear wavenumber β by the following equation.

$$\begin{aligned} n_\phi &= \frac{\partial\theta}{\partial z}\cdot\frac{\partial z}{\partial\phi} \\ &= \{r_b + \eta\, cos(\theta_c)\}.\beta \end{aligned}$$

If $\tilde{w}_\phi = \partial\phi/\partial t$, then the rectilinear velocity $r\tilde{u}^{\xi^3} = \{r_b + \eta cos\theta_c\}\tilde{w}_\phi$ decays exponentially in η with the same rate as $\tilde{w}$. Hence, we can derive the following differential equation for $\boldsymbol{f} = \{\tilde{T}, \tilde{u}^{\xi^1}, \tilde{u}^{\xi^2}, \tilde{w}_\phi\}^T$ to be applied as a boundary condition.

$$\begin{aligned} \frac{\partial \boldsymbol{f}}{\partial\eta} + \boldsymbol{D}\boldsymbol{f} &= 0 \\ \boldsymbol{D} &= \left\{\sqrt{(\alpha^2+\beta^2)(1-\overline{M_e}^2)} + \frac{C_\omega}{r_b + \eta\, cos\theta_c}\right\} \boldsymbol{I} \end{aligned}$$

where $\boldsymbol{I}$ is a 4×4 identity matrix and C_ω is 1 for $f^4 = \tilde{w}_\phi$ and is zero otherwise. The equation for $\boldsymbol{f}$ may be transformed into that for $\tilde{\boldsymbol{Q}}$ other desired variables in the insert file `boundary.ins`. It is discretized using hermitian finite-difference formulae and applied at the boundary truncated at $\eta_e \sim 1.5 - 1.8\delta_{bl}$ where δ_{bl} is the boundary-layer thickness. For the hypersonic boundary layer over the cone, gradients of basic-state variables with respect to η are small at this location. The error in the boundary conditions caused by the nonuniformity of the freestream in η should be small as it is scaled by a factor of $O\{\frac{1}{r_b+\eta cos\theta_c} e^{-\boldsymbol{D}\eta_e}\}$.

Appendix C

Bstate.ins for Hypersonic Cone Applications

The *bstate* file which reads the FORTRAN unformatted PLOT3D planes files or the TLNS solution of Esfahanian (1991) is complicated because it interpolates a basic state solution defined numerically on a grid. It currently assumes that the grid is structured, but the number of grid points along the coordinate lines passing through the body surface is allowed to vary from station to station (as it does in the Kaups & Cebeci (1977) boundary layer code). Moreover, the *bstate* file currently uses tensor product interpolation (i.e., compositions of one-dimensional interpolants) and works only when the transformation between the basic state coordinates (K, I, J) and the disturbance coordinates (ξ^1, ξ^2, ξ^3) is of the form:

$$\xi^1 = \xi^1(K) \tag{C.1}$$
$$\xi^2 = \xi^2(K, I, J) \tag{C.2}$$
$$\xi^3 = \xi^3(J) \tag{C.3}$$

Some minor modifications would enable the transformation to be of the form:

$$\xi^1 = \xi^1(K, J) \tag{C.4}$$
$$\xi^2 = \xi^2(K, I, J) \tag{C.5}$$
$$\xi^3 = \xi^3(K, J) \tag{C.6}$$

These modifications have not yet been implemented. More extensive modifications would be required to enable the transformation to be arbitrary.
The restriction that ξ^1 and ξ^3 be independent of I means that the ξ^2 coordinate lines must have the same *shape* and *direction* as the I coordinate lines. However, each ξ^2 coordinate line is labeled with the values of (ξ^1, ξ^3) which are constant along it instead of the values (K, J) attached to the corresponding I coordinate line. Hence, if the I coordinate lines are all straight and *perpendicular* to the body surface, so must be the ξ^2 coordinate lines. If the I coordinate lines are all straight and *oblique* to the

surface, so must be the ξ^2 coordinate lines. Etc. This restriction is not severe because the stability codes can use the same nonorthogonal coordinates that modern basic state codes use. Also, to make the boundary layer approximation or the thin layer approximation, the basic state viscous flow solvers use grids with the I coordinate lines orthogonal or nearly orthogonal to the body surface. Hence, no adverse effects on the accuracy of the PSE approximation should be expected.

The interpolation that `bstate` itself is responsible for is primarily along the ξ^2 coordinate lines. Another subroutine, `mdintr`, determines the stations and interpolants for interpolating the basic state in ξ^1 and ξ^3. (See below.) First, `bstate` calls `mdintr` . Next, it distributes the points `chy(j)` along each ξ^2 coordinate line specified by `mdintr` and computes the corresponding values of the basic state using Lagrange polynomials as interpolants. It then differentiates the Lagrange interpolants with respect to ξ^2 to get the ξ^2 derivatives of the basic state at the same stations and points. Finally, knowing the data at the proper ξ^2 points but at surrounding ξ^1 and ξ^3 stations, `bstate` interpolates the profiles and their ξ^2 derivatives onto the desired station using the interpolants specified by `mdintr` . The ξ^1 and ξ^3 derivatives of the basic state are then obtained from the corresponding derivatives of the (ξ^1, ξ^3) interpolants.

It is important to note that subroutine `bstate` uses the *same* grid points `chy(j)` at all the stations specified by `mdintr` even when `pointd.ins` makes the disturbance grid ξ^2 point distribution vary from station to station. This makes it easier to numerically evaluate the derivatives of the basic state with respect to ξ^1 and ξ^3 holding ξ^2 *constant* . The vector `chy(j)` may, however, be different the next time that `bstate` is called but again will be locally held constant during the interpolation.

Subroutine `bstate` reads the data it needs for interpolation from a direct access file. This data access method enables `bstate` to go directly to the appropriate place in the file and read in the desired point of any station profile without having to first read in all of the points in the preceding profiles. The direct access method also enables `bstate` to read the profiles in any order that the user specifies, even from last to first. Hence, the stability investigation can begin at some downstream location where it is often easier to find unstable eigenvalues. From there, the analysis can proceed upstream by continuation into the stable region.

The FORTRAN unformatted direct access file is set up by subroutine `gridbs` through calls to a number of additional routines[1]. These additional routines are: `bsname`, `bsiois`, `bdiois`, `bpstda`, and `scale`.

Two of these routines are independent of the basic state *input* file formats and are stored in file *util.f*. They are: `bsname` and `bdiois`. *util.f* also contains subroutine `mdintr` because `mdintr` should not need to be modified for different flow solver interfaces. None of the routines in *util.f* use any variables in `common`.

Subroutines `bsiois`, `bpstda`, and `scale` are stored together in file *aux.f* and may need

[1]The direct access file would not really have to be created if each input file itself were direct access, but this is rarely the case.

to be changed when developing a new interface to a different basic state flow solver. We have tried to circumvent the need for extensive revisions whenever possible by developing an interface to the popular PLOT3D format. Nevertheless, not every flow solver creates PLOT3D files.
The additional routines that `bstate` and `gridbs` use to help with the data interpolation and file management are described in the following subsections.

C.1 mdintr

As mentioned above, subroutine `mdintr` (MultiDimensional INTeRpolation) determines which stations to use for interpolating the basic state solution in the variables ξ^1 and ξ^3. It also computes the Lagrange interpolants and their derivatives with respect to ξ^1 and ξ^3.
The Lagrange interpolants are products of polynomials of orders `ngk` and `ngj` in ξ^1 and ξ^3, respectively. Tentative values of `ngk` and `ngj` are passed as arguments to `mdintr` . However, the input `ngk` and `ngj` may be changed if they exceed the corresponding basic state grid dimensions or the bounds of the memory allocated for the interpolants.
Either one or two-dimensional interpolation may be required depending upon the dimensions of the basic state solution.
One-dimensional interpolation with Lagrange polynomials will be used in the direction for which the grid dimension is greater than one. The interpolant in the other direction (with grid dimension 1) will be set to 1. Only a one-dimensional search scheme is needed in this case.
Two-dimensional interpolation with Lagrange polynomials will be used whenever both grid dimensions are greater than one. In this case, a two-dimensional search scheme will be used to find the appropriate place in the basic state grid. The scheme implemented here uses a variant of Newton-Raphson iteration to solve the equations:

$$\xi_p^1 = \xi^1(RK, RJ) \tag{C.7}$$
$$\xi_p^3 = \xi^3(RK, RJ) \tag{C.8}$$

for the basic state grid indices (RK, RJ) corresponding to the point (ξ_p^1, ξ_p^3). *This scheme may not always work, even when the Jacobian is never zero.* However, it is fast and in practice has been reasonably robust.
Once the real values RK and RJ have been determined, neighboring stations (ξ_k^1, ξ_j^3), `k=1,2,...,ngk`, `j=1,2,...,ngj` are selected for the interpolation. Tentatively, `nbk=ngk/2` points below and `ngk-nbk` points above ξ_p^1 in the basic state grid are used for interpolating in ξ^1:

$$\xi_k^1 = \xi^1(int(RK) - nbk + (k-1)) \tag{C.9}$$

where $\int(RK)$ is the integer obtained by eliminating the fractional part of RK. Similarly, `nbj=ngj/2` below and `ngj-nbj` points above ξ_p^3 are used for interpolating in ξ^3:

$$\xi_j^3 = \xi^3(int(RJ) - nbj + (j-1)) \tag{C.10}$$

The `stndat` addresses of all these stations are stored in the integer vector `ida(1:nsrf)` . The total number of points, `nsrf = ngk*ngj`, is later output by subroutine `mdintr` .

After determining which stations to use for the interpolation, subroutine `mdintr` computes the interpolating polynomials. These interpolants are taken to be polynomials in (ξ^1, ξ^3) instead of (K, J) because the basic state flow usually varies smoothly with (ξ^1, ξ^3) while it may not with (K, J). Basic state grid spacing discontinuities which are often present[1] cause discontinuities in the derivatives of the flow with respect to (K, J). Such discontinuities make it more difficult to use higher order interpolants.

To ensure that the interpolants in (ξ^1, ξ^3) are well defined[2], the transformation from the basic state coordinates (K, J) to the disturbance coordinates (ξ^1, ξ^3) is restricted to be of the form:

$$\xi^1 = \xi^1(K), K = 1, 2, \ldots, \texttt{kdim} \tag{C.11}$$

$$\xi^3 = \xi^3(J), J = 1, 2, \ldots, \texttt{jdim} \tag{C.12}$$

where `kdim` and `jdim` are the dimensions of the basic state grid in the K and J directions, respectively. This restriction can be eliminated by interpolating, say, in the arc length along the basic state grid lines. However, the restriction has not yet been removed.

The use of the rectilinear grid in (ξ^1, ξ^3), equations (C.11) and (C.12) above, enables the multi-dimensional interpolation to be done by composing two one-dimensional interpolants:

$$F(\xi^1, \xi^3) = \sum_{j=1}^{ngj} F(\xi^1, \xi_j^3)\phi_j(\xi^3) \tag{C.13}$$

$$F(\xi^1, \xi_j^3) = \sum_{k=1}^{ngk} F(\xi_k^1, \xi_j^3)\psi_k(\xi^1) \tag{C.14}$$

or

$$F(\xi^1, \xi^3) = \sum_{j=1}^{ngj}\sum_{k=1}^{ngk} F(\xi_k^1, \xi_j^3)\psi_k(\xi_k^1)\phi_j(\xi^3) \tag{C.15}$$

[1] especially in the files which the AFWAL PNS code creates because of the seemingly arbitrary way in which it prints out data

[2] See, for instance, Lancaster & Salkauskas (1988), pp.135-137.

where F is an arbitrary function and

$$\phi_j(\xi^3) = \frac{\prod_{i=1,i\neq j}^{ngj}(\xi^3 - \xi_i^3)}{\prod_{i=1,i\neq j}^{ngj}(\xi_j^3 - \xi_i^3)} \tag{C.16}$$

$$\psi_k(\xi^1) = \frac{\prod_{i=1,i\neq k}^{ngk}(\xi^1 - \xi_i^1)}{\prod_{i=1,i\neq k}^{ngk}(\xi_k^1 - \xi_i^1)} \tag{C.17}$$

The values of $\psi_k(\xi^1)$ evaluated at ξ_p^1 are stored in `bik(0,k)` . The `ndk` derivative of $\psi_k(\xi^1)$ with respect to ξ^1, evaluated at ξ_p^1, is stored in `bik(ndk,k)` . Similarly, the `ndk` derivative of $\phi_j(\xi^3)$ with respect to ξ^3, evaluated at ξ_p^3, is stored in `bij(ndj,j)` . The order `ndmx` of the highest derivative, mixed or otherwise, is passed as an input to `mdintr` . The derivative of the tensor product interpolant $\psi_k(\xi_p^1)\phi_j(\xi_p^3)$ with respect to ξ^1 `ndk` times and ξ^3 `ndj` times is stored in the matrix `bi(idad,ia)` , where `ia = ngj*(k-1)+j` and `idad = ndj*(2*ndmx-ndj+3)/2+ndk` for $0 \leq$ `ndj` $\leq$ `ndmx` and $0 \leq$ `ndk` $\leq$ `ndmx-ndj`
All of the station data is interpolated and numerically differentiated using the matrix `bi` . If the body coordinates are stored in `stndat`, this computes the body slopes, body curvature, etc. Control is then returned to the calling program.

C.2 bsname

Subroutine `bsname` determines the names `bsfn(n)` `(n=1,2,...,nbsf)` of the `nbsf` basic state input files. It also creates a corresponding direct access file name for each basic state input file. The other subroutines which read and write the basic state information may then use as many direct access files as there are basic state input files. However, the present scheme stores all of the information in one direct access file only: the first one.
The algorithm that `bsname` uses to determine the name `bsfn(n)` of input file `n` is as follows. First, it checks to see if the file `fort.un` exists. Here, `un = ubsf(n)` is the (integer) unit that file number `n` will be connected to for input. If `fort.un` exists, then `bsname` assigns `bsfn(n) = fort.un` . The user can enable this by simply soft-linking the input files to the corresponding files `fort.un` . If `fort.un` does *not* exist, then `bsname` prompts the user with a message to input the corresponding file name. The message is: `''Enter the name of the basic state '', messag(1b:le),'' file > ''`, where `messag(1b:le)` is the n^{th} substring ending with a semicolon delimiter in the character variable `messag` passed to `bsname` by `gridbs`[1] .
Once `bsname` has determined `bsfn(n)`, it assigns a corresponding name to `dafn(n)` using the following scheme. If `bsfn(n) = fort.ubsf(n)`, then `bsname` assigns `dafn(n) = fort.uda`[1] . If `bsfn(n)` $\neq$ `fort.ubsf(n)`, then `bsname` strips off any extension

[1]i.e., if `messag` = 'grid;solution;parameters;', then the second message is "Enter the name of the basic state solution file > ".

[1]The file extension "`uda`" stands for "unformatted direct access".

following "." in the file name bsfn(n) and replaces it with "uda" . If bsfn(n) does *not* have an extension, then bsname adds the extension ".uda" . In any of these cases, the extension for each file might not be unique. To avoid this, *the base name of* bsfn *must be unique in order for the corresponding name* dafn *to be unique* . This is not really important in the present version of the stability codes because they only use one direct access file anyway.
The arguments of subroutine bsname are: nbsf, ubsf,uda,bsfn,dafn, and messag . ubsf and uda are integer vectors of length nbsf, while bsfn and dafn are character vectors of length nbsf . messag is a character variable with nbsf substrings, each substring ending in a semicolon delimiter. bsname returns to the calling routine after determining all of the names of the files. It does not itself open or close any files.

C.3 bsiois

Subroutine bsiois reads into core memory all the basic state data which depend only on ξ^1 and ξ^3. Examples of such data are: number of points in the profile; body radius; shock stand-off distance, etc. All of this data is stored in the array stndat(0:nstpar,mxnsta), where nstpar is the number of station parameters and mxnsta is the maximum number of stations. To utilize this information, the interpolation routines all require that it be recorded in the following elements of stndat:

stndat(0,k) = the cumulative number of points in all the profiles preceding station k, plus one. The user must have bsiois compute this number. Subroutine bdiois will later add to stndat(0,k) and *replace* it with the direct access file record number for the beginning of profile k after determining how much space is required for all of the problem specific data and station data.

stndat(1,k) = the value of ξ^1 for station k . Must be set by subroutine bsiois after being read or computed from the basic state input files.

stndat(2,k) = the number of ξ^2 grid points in the profiles at station number k . Must be set by subroutine bsiois after being read from the basic state input files.

stndat(3,k) = the value of ξ^3 for station k . Must be set by subroutine bsiois after being read or computed from the basic state input files.

Additional information such as the Cartesian coordinates, body radius, or shock stand-off distance may also be stored in `stndat` to be used by `metric` or `btrans` .
The assignment of the disturbance coordinates to the basic state grid points is vital because it enables the interpolation routines to find their way around in the basic state grid. This may require that `bsiois` transform the basic state coordinates (K, J) to the disturbance coordinates (ξ^1, ξ^3) if they are not the same. The transformation of I to ξ^2 must be made by subroutine `bpstda`, described below, as it reads in the profiles and writes them out to the direct access file.
Subroutine `bsiois` also assigns problem specific data to one-dimensional vectors. There is one vector available for each data type expected: `logicv(0:nlp)` for logical, `charv(0:ncp)` for character, `intv(-6:nip)` for integer, and `realv(0:nrp)` for real. The logical vector may contain flags which are set by the basic state flow solver. The character vector typically contains some lines of a title. The integer vector contains the basic state grid dimensions. Finally, the real vector usually contains the reference temperature and free-stream Mach number, etc. In order to access this data, the other routines require that `bsiois` make the following assignments.

integer

`intv(-6)` = `nip`, number of integer variables.

`intv(-5)` = `nlp`, number of logical variables.

`intv(-4)` = `ncp`, number of character variables.

`intv(-3)` = length of character variables.

`intv(-2)` = `nrp`, number of real variables.

`intv(-1)` = number of station parameters.

`intv( 0)` = number of basic state variables.

`intv( 1)` = K dimension of basic state grid.

`intv( 2)` = total number of stations in file.

`intv( 3)` = J dimension of basic state grid.

real

`realv(1)` = ξ^1 scale of basic state grid.

`realv(2)` = ξ^2 scale of basic state grid.

`realv(3)` = ξ^3 scale of basic state grid.

`realv(4)` = $\xi^4 = \tau = t$ scale of basic state grid.

`realv(4+k)` = scale of basic state solution variable number `k`, `k=1,2,...,nbsvar`, where `nbsvar` = `intv(0)` is the number of basic state variables.

C.4 scale

After all of the station data has been read in from the original basic state input files, subroutine `gridbs` calls `scale` . Subroutine `scale` must assigns the scales, `scaldg(1:4)` and `scaldv(1:5)`, respectively, for the disturbance coordinates $(\xi^1, \xi^2, \xi^3, \xi^4)$ and default disturbance variables (ρ, T, u^1, u^2, u^3). Here, u^i is the i^{th} velocity component in inertial Cartesian coordinates. `scale` must also set the *ratios* `scalbg(1:4)` of: the disturbance coordinate scales used by the basic state solution to: the corresponding scales assigned for the stability analysis. Finally, it also must set the ratios `scalbv(1:nbsvar)` of: the basic state input variables to: the corresponding scales assigned for the stability analysis. The basic state scales set in subroutine `bsiois` and stored in the direct access file are usually used here[1]. The ratios are later used by subroutines `mdintr` and `bstate` to properly rescale the basic state input for the stability analysis.

C.5 bdiois

The first time that subroutine `gridbs` calls `bdiois`, `bdiois` copies all of the station data and problem specific data to the top of the direct access file. Subroutine `gridbs` calls `bdiois` *prior* to subroutine `bpstda` (see below) because `bdiois` computes `stndat(0,k), k=1,2,...,intv(2)`, the direct access file record number for the beginning of each basic state profile [2]
The next time that the program is run, the direct access file will already exist. In this case, subroutine `gridbs` will still call `bsname` to get the name(s) of the direct access file(s). However, `gridbs` need not call subroutines `bsiois` and `bpstda` to get the stations data and profiles. Instead, subroutine `gridbs` will call `bdiois` to *read* the station data from the direct access file. Subroutine `bstate` will, as before, read the profiles from the direct access file.

C.6 bpstda

Subroutine `bpstda` copies all of the basic state input profiles to the direct access file. All of this data must be scaled as specified by the assignment statements to `realv` in subroutine `bsiois` . The profiles must be written to the unformatted direct access file using statements similar to the following:

```
do 20 k=1,intv(2)
```

[1]When a particular variable really has *no* scale, its scale can be set arbitrarily *as long as the scale is consistent with the dimensionless parameters* .

[2]The record number is computed and stored before the array `stndat` is copied to the direct access file.

```
        do 10 n=1,nint(stndat(2,k))
C
C Read in the basic state variables qb(nbsv) at point n in profile k.
C Compute ξ² at point n in profile k.
C
           write(uda(1),rec=key+n-1) ξ²,(qb(m),m=1,nbsv)
10       continue
20     continue
```

where `intv(2)` is the number of basic state profiles, `stndat(2,k)` is the number of points in profile `k`, and `nbsv = intv(0)` is the number of basic state variables. Again, the integer vector `intv` must be previously set in subroutine `bsiois` .
Once the direct access file `dafn(1)` has been created, the input basic state files may be closed. The only routines that will subsequently execute will be the interpolation routines `mdintr` and `bstate` . They will only use the information in the direct access file.